

MPEX / 3000

User Manual

(Version 3.3)

Copyright (C) 1980–2017

**Written by Eugene Volokh
Vice President, Research and Development**

**VESOFT, Inc.
9213 Warbler Place,
Los Angeles, CA 90069. USA
Phone (310) 282-0420
Fax (310) 785-9566**

Disclaimer

The information in this document is subject to change without notice. VESOPT, Inc. makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of MERCHANTABILITY and FITNESS FOR A PARTICULAR PURPOSE.

VESOPT, Inc. shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

You may obtain additional copies of the manual directly from VESOPT or VESOPT's representatives.

Formatting

This manual was originally formatted on the HP3000 using PROSE, Robelle Solution Technology Inc's excellent text formatter.

Corrections applied and Word/PDF conversion performed by Paul Taffel, using Bitstream Charter, Source Sans Pro, and Source Code Pro open-source fonts. Updated August 2017.

TABLE OF CONTENTS

INTRODUCTION.....	1
Examples of MPEX commands	2
System management	2
Managing disk space	3
Program development.....	4
Database management.....	5
File system security	6
Job stream programming.....	6
RUNNING MPEX.....	8
Running MPEX interactively.....	8
Running MPEX in batch	9
Running MPEX with ;INFO=	10
Specifying commands to be executed each time MPEX is entered	10
Important security considerations for the MPEXMGR file.....	12
Speeding up MPEX start-up.....	13
CI.PUB.VESOPT (for MPEX/iX CI emulation)	15
USING MPEX.....	16
Changing the MPEX prompt.....	16
Running under MPEX/iX	16
Entering MPE commands in MPEX (':' and 'MPE' prefixes).....	17
YES/NO prompting ('?' prefix)	18
Batch execution ('\$' prefix)	19
Implied run	21
Which program file is run?	22
Changing the search path	22
Specifying ;INFO= and ;PARM=	23
Command files	23
Special security note	25
Command files and "HPPATH"	25
ANYPARM and REST\$ parameters.....	25
CS\$ parameters.....	27
Determining whether you're in MPEX or not	27
Redirecting command input and output.....	28
Checking MPEX command success	29
Setting default parms for MPEX commands	30
Restricting MPEX commands to only work on certain files	31
MPEX VARIABLES	32
Using MPEX variables.....	32
Setting MPEX variables	34
Local variables	34
Other variable features.....	35
A few comments about the "!" character	37
Special variables used by MPEX	38
FILENAMES AND FILESETS.....	41
MPE filenames.....	41
MPE filesets	41

CONTENTS

POSIX filenames	42
POSIX filesets	43
MPEX filesets	43
+ filesets	45
- filesets	45
Indirect files	46
Selection criteria	47
File ranges	48
Target filesets	50
POSIX Note: source and target filesets using HFS syntax	51
File attribute variables and functions	52
File attributes pertaining to IMAGE datasets	61
File attributes pertaining to programs	62
File attributes pertaining to SPOOL files	63
GROUP and ACCOUNT attributes of files	64
JOB/SESSION attributes	65
SECURITY/3000 user profile attributes	66
SPECIAL MPEX TOPICS:	67
Using MPEX to compile programs	67
GENERAL SYNTAX:	67
Compiling many sources into one program	68
Compiling without :PREPping	69
Compiling with a COPYLIB	69
Specific differences between MPE and MPEX compiler commands	69
Using third party compilers with MPEX	70
MPEX HOOK	71
Executing MPEX and MPE commands, and UDCs from EDITOR, QUERY or any other utility	71
REDOing subsystem commands	73
Activating the father process	74
Building files in other accounts	75
MPEX HOOK preserves ACDs in EDITOR, TDP, etc.	76
How to %HOOK a program	77
How to run a %HOOKed program	78
MPEX process handling	78
Managing suspended son processes	79
HOOKed programs	79
A cautionary note	80
Passing input to son processes	81
:WAKE, MPEX's improvement to [BREAK] and :ABORT/:RESUME	82
%RUN...;PRI=...	84
%RUN...;STDIN=...;STDLIST=...	84
%RUN...;GOON	84
%RUN...;NOACTIVATE, ;NEW, and ;KILL	85
The MPEXPIN JCW	86
MPEX and your system security	86
Removing "creator-only" restrictions	86
Copying files into other accounts	87
Working on databases	87
Spool files	87
Lockwords	88
GOD — gives SM capability and :ALLOWs all commands	89
MORTAL — undoes a previous "GOD" operation	90
VEMODIFY — line editing made easier	90
Callable VEMODIFY function for expression programs and command files	91

Online notepad.....	93
Use the ; prefix to enter a note	93
Searching and printing notepad entries.....	94
Editing or deleting notepad entries.....	94
Multiple notepads — a mini-file cabinet.....	95
Notepad file names – naming using VESOFTNOTEFILE variable.....	95
Changing the notepad prefix character (;).....	95
MPEX COMMAND REFERENCE.....	96
%ABORTJOB	98
ABORTJOB and STREAMX scheduled jobs	99
%ALARM, %SHOWALARM, %DELETEALARM	99
Repetitive alarms	101
Security considerations	101
Performance considerations	102
Viewing pending alarms (%SHOWALARM)	102
Deleting pending alarms (%DELETEALARM)	103
%ALIAS, %UNALIAS	103
%ALLOCATE	104
%ALTFILE.....	105
POSIX note: %ALTFILE security considerations	106
Changing file blocking factors (;BLKFACT=)	106
Saving disk space	106
Speeding up file access.....	107
WARNING	108
Changing program file capabilities (;CAP=).....	108
Related MPEX features	109
Changing file codes (;CODE=)	110
Changing file ownership (;CREATOR= and ;GROUPID=)	111
Related POSIX features	111
Related MPEX features	113
Moving files from disk to disk (;DEV=)	113
What you should know about extents.....	114
What you should know about non-system volumes	115
Changing file maximum # of extents (;EXTENTS=)	116
Changing file flimits (;FLIMIT=)	118
Saving disk space	118
What if a file overflows?.....	118
WARNING	120
Changing file initially allocated extents (;INITEXTENTS)	121
Preventing changes to file access/modify dates (;KEEPAMDATES)	121
Changing file lockwords (;LOCKWORD=)	122
Security considerations	123
Related MPEX features	123
Changing program file maxdata value (;MAXDATA=)	123
Related MPEX features	124
Setting file flimits = to their eofs (;SQUEEZE).....	124
Changing program file stack value (;STACK=)	124
Releasing file unused disk space on MPE/iX (;XLTRIM)	124
Changing the internal structure of a file	125
%ALTJOB	126
%ALTPROC	126
%ALTSCHED	126
%ALTSEC	128
%ALTSPoolFILE	129
MPE/iX native mode spooler note:	130

CONTENTS

%BACKG	130
%BACKG STARTJOB (requires SM or OP capability)	131
%BACKG STOPJOB	131
%BACKG START, <i>taskname</i>	132
%BACKG STOP, <i>taskname</i>	132
%BACKG SHOW	132
BACKGINI.DATA configuration file	133
Setting execution priority for BACKG tasks	133
BACKG tasks	133
%BREAKJOB	135
%BYE	135
%CALC	136
%CALENDAR	137
%CHGROUP	137
%CHLOGON	138
Eliminating password prompts (\$CHLOGON-NOPASS)	140
Restricting who may use %CHLOGON (\$CHLOGON-FORBID PERMIT)	141
Using abbreviated logons with %CHLOGON	142
Important note for MPE/iX users	142
%COPY	143
What if the "to-file" already exists? (;YES NO ASK)	144
How is the "to-file" built?	144
Special note on private volumes	145
Copying into other accounts	145
Copying databases	146
Copying KSAM files (;KEYFILE=)	146
Copying to other computers (;DEV= <i>envid</i> ..)	147
Preserving last access/last modify dates (;KEEPAMDATES)	148
Preserving ACDs (;COPYACD)	148
The %COPY command's speed	149
Copying single files	149
Lockwords and %COPY	149
Copying data into existing files	150
%DBxxxALT commands	150
%DBADGALT	151
%DBGENALT	151
%DEALLOCATE	151
%DELETEALARM	152
%DELETESPOOLFILE	152
MPE/iX native mode spooler note:	153
%DELETELVAR	153
%DELETEVAR	153
%DEMO	153
%DEVCONTROL	154
%DO	154
%DOSAVED	154
%ECHO	155
%EDIT	155
When will %EDIT "/KEEP" the file?	156
%EDITCHG	156
%EDITQUAD	157
%EDITTDP	158
%ERASE	158
%ERRCLEAR	159
%ESCAPE	159

%EXIT	159
%FCOPY	159
%FILTER	160
%GOON, %SHOWGOON	161
Checking status of %GOON processes (%SHOWGOON)	162
Setting %GOON process priority.....	162
%HELP	162
Getting help on SECURITY and VEAUDIT topics	164
Getting help on MPE CI, File, and Loader errors.....	164
%HELPMAKE.....	165
%HOOK.....	166
%IDENTIFY	166
Security considerations	167
%IF...%ELSEIF...%ELSE...%ENDIF	167
%INITUDCS.....	168
%INPUT.....	169
%KILL.....	170
%LISTF.....	171
%LISTF modes	172
%LISTF...,0 [FILES]	173
%LISTF...,1 [SUMMARY]	174
%LISTF...,2 [DISC]	174
%LISTF...,3	175
%LISTF...,4	177
What is the extent map?.....	177
Related MPEX features	178
%LISTF...,5	178
Related MPEX features	179
%LISTF...,6 [FILENAME, QUALIFY].....	179
%LISTF..., -1 [LABEL]	180
%LISTF..., -2 [ACD]	180
%LISTF..., -3	181
%LISTF...,ACCESS	182
Databases.....	183
Performance notes.....	183
%LISTF...,DATES	184
%LISTF...,DB.....	185
Related MPEX features	186
Notes on DBOPEN modes	186
%LISTF...,DISCUSE.....	187
%LISTF...,ID	188
%LISTF...,SAVABLE	189
%LISTF...,SEC	190
Related MPEX features	191
%LISTF...,POSIX	192
%LISTF...,XL3 [DETAIL]	192
%LISTF...,XL4 [SECURITY].....	193
%LISTF — defining your own %LISTF formats	194
The full syntax of a flexible LISTF format file	198
%LISTJOB	202
%LISTREDO	204
%MANY	204
%MPEXSTORE	205
%NEWLINK	206
%NOMSG	207
%OPTION.....	207

CONTENTS

%PAUSE.....	208
%PAUSEJOB, %SHOWPAUSED	208
Performance considerations (;CHECKEVERY=)	209
Viewing "paused" jobs (%SHOWPAUSED)	209
%PRINT.....	210
%PRINT keywords	212
Searching for strings (;SEARCH=)	217
String searches in fileset selections	220
%PRINTI, %PRINTO.....	221
Using ;WAIT, ;IFLOW, & ;SEARCH=... to view \$STDLISTS	221
%PROGINFO	222
%PURGE.....	222
Automatic verification to avoid mistakes	223
%PURGELINK.....	225
%QEDIT	226
How it works.....	226
%QUIT	227
%REDO, %DO, %LISTREDO	227
Referring to saved commands.....	229
Editing the command being %REDOne.....	230
Abbreviations for %REDO, %DO and %LISTREDO	231
%LISTREDO parameters	232
Saving your command history as a permanent file	233
Setting MPEX prompt to make %REDOing easier	233
:REDO, :DO and :LISTREDO in other programs	234
%REDO facility options.....	234
%RELEASE	235
%REMOTE HELLO	236
%RENAME	237
Lockwords and %RENAME	239
%REPEAT...%FOREACH.....	240
%REPEAT...%FORFILES.....	240
Accessing file information	241
"From" filesets and "To" filesets.....	242
Running programs and passing them input	243
Operating on spool filesets.....	243
Operating on temporary filesets.....	244
%REPEAT...%FORJOBS	245
%REPEAT...%FORNUM.....	245
%REPEAT...%FORPROFILES.....	246
%REPEAT...%FORRECS	246
%RESUMEJOB	247
%RETURN	247
%RUN.....	248
%RUNCREATE, %RUNINPUT, %RUNACTIVATE	249
%SAVEJOB, %SHOWSAVED, %DOSAVED	250
Viewing saved jobs (%SHOWSAVED)	252
Re-submitting saved jobs (%DOSAVED)	252
%SCHEDULE	253
%SEC.....	255
%SECURE.....	255
Vital for system security!	256
%SET CAPABILITY	256
%SET CREATORPROTECT	257
%SET DATE.....	258

%SET DEFAULT	259
%SET GOONMAXPRI	260
%SET [NO]CMDTRACECHECK.....	260
%SET [NO]VARTRACE	261
%SETLVAR	262
%SETVAR.....	262
%SHOW	263
%SHOWALARM	263
%SHOWGOON	263
%SHOWJOB	264
%SHOWLVAR.....	265
%SHOWME	265
%SHOWOUT	266
MPE/iX Native Mode Spooler note	267
%SHOWOUTJ	267
%SHOWPAUSED.....	268
%SHOWPROC	268
Showing which files a process has open (;FORMAT=FILES).....	269
%SHOWSAVED	270
%SHOWTREE.....	270
%SHOWVAR	271
%SPOONFEED	272
How to use spoonfeeding.....	273
An important warning.....	275
Getting information on which programs you're spoonfeeding	275
%SPOONINIT	275
%SUBMIT	276
%SYNTAX.....	278
%TELL	278
%TRAPERROR	279
%CLEANUP	280
%UNALIAS	281
%VEAUDIT	282
%VECMDCH.....	282
%VEOPENCH	282
%WARN	283
%WARNF	283
%WHEREIS	284
%WHILE...%ENDWHILE	285
%WITHCAPS	286
%XEQ	289
APPENDIX A: POSIX COMPATIBILITY ISSUES.....	A-1
Important note for users of MPEX/iX 4.5:	A-1
Visual VEMODIFY (for %REDO) extends over 1 screen line	A-1
Using the continue ("&") character.....	A-1
"@.@.@ " vs. "/" on systems that support POSIX	A-2
LISTF format changes.....	A-2
All %LISTF modes:.....	A-2
%LISTF ,-1:	A-3
%LISTF ,2:	A-3
%LISTF ,-3 and %LISTF ,3:	A-3
File attributes: CREATOR, CREATORACCOUNT, and OWNER.....	A-3
Extended filesets using CREATOR= and CODE=	A-4
Errors and warnings in STDLIST(...) honor I/O redirection.....	A-4

CONTENTS

ISxxxCAP variables/functions fixed.....	A-5
%ABORTJOB sched can abort STREAMX scheduled jobs.....	A-5
%ALARM starts the BACKG job if it's not already running.....	A-5
%ALTFILE program;CAP=-xx,yy,zz changed.....	A-5
%ALTFILE ;CREATOR changes on POSIX systems.....	A-6
%SET CREATORPROTECT now enforced on POSIX systems.....	A-6
VEFxxx functions, changes for SM users.....	A-6
MPE error messages down-shifted.....	A-7
Expressions: TYPEOF() enhanced for special types.....	A-7
QEDIT "jumbo" files supported by %PRINT & %LISTF,ID.....	A-7
HPREDOSIZE variable changed to MPEXREDOSIZE.....	A-7
APPENDIX B: VESOF EXPRESSIONS.....	B-1
Introduction	B-1
Variables.....	B-1
Data types	B-2
Integer and boolean operators and functions.....	B-4
String operators and functions	B-7
Date operators and functions	B-11
Real operators and functions.....	B-13
Time operators and functions.....	B-14
Special MPE access functions	B-16
HP terminal manipulation functions	B-19
SECURITY-related functions.....	B-20
File attribute functions	B-21
FINFO Function Attributes.....	B-23
FILE I/O functions.....	B-26
Database I/O functions.....	B-30
Opening a database for access	B-30
Determining the status of a VEDBxxx call	B-30
Trapping program output (\$STDLIST)	B-34
Terminal input	B-35
Terminal input: VEMODIFY	B-36
Output formatting	B-37
String formatting	B-37
Integer formatting.....	B-38
Date formatting.....	B-39
Time formatting.....	B-40
Real number formatting.....	B-40
Advanced expression programming	B-41
IF ... THEN ... ELSE	B-41
WHILE ... DO	B-42
FOR <i>stringvar</i> IN FILE (<i>fopenparms</i>) DO	B-43
TRY ... RECOVER and TRY ... CLEANUP.....	B-44
Expression program variables	B-44
Outputting data	B-46
Comments.....	B-47
Writing expression programs.....	B-47
A very important disclaimer.....	B-49
APPENDIX C: LOADER ERROR MESSAGES EXPLAINED.....	C-1
APPENDIX D: CRYPTIC FILE SYSTEM ERROR MESSAGES DECRYPTED.....	D-1

Printing Conventions

This section describes the various conventions used to describe MPEX ommand syntax.

Notation	Usage and examples
UPPERCASE	<p>Within syntax descriptions, characters in uppercase must be entered in exactly the order shown, though you can enter them in either uppercase or lowercase.</p> <p>For example, valid variants of SHOWJOB command:</p> <pre>ShowJob Showjob SHOWJOB</pre> <p>and invalid variants:</p> <pre>Shojwob ShoJob SHOW JOB</pre>
<i>Italics</i>	<p>Within syntax descriptions, a word in italics represents a formal parameter or argument that you must replace with an actual value. In the following example, you must replace filename with the name of the file that you want to release:</p> <pre>RELEASE <i>filename</i></pre>
Punctuation	<p>Within syntax descriptions, punctuation characters (other than brackets, braces, vertical lines, and ellipses) must be entered exactly as shown.</p> <p>Note that the "%" shown at the beginning of any command should NOT be entered as part of the command, it is simply being included to remind you that this is an enhanced MPEX command and not a normal MPE command.</p>
{ }	Within syntax descriptions, braces enclose required elements.
{item item}	<p>When several elements are separated by vertical lines, you must select one and only one of the items specified. In the following example, you must select on or off:</p> <pre>SETMSG { ON OFF }</pre>
[]	<p>Within syntax descriptions, brackets enclose optional elements. In the following example, brackets around ,TEMP indicate that the parameter and its delimiter are optional:</p> <pre>PURGE <i>filename</i> [,TEMP]</pre>
[...]	<p>Within syntax descriptions, a horizontal ellipsis enclosed in brackets indicates that you can repeatedly select elements that appear within the immediately preceding pair of brackets or braces. In the following example, you can</p>

CONVENTIONS

select *itemname* and its delimiter zero or more times. Each instance of *itemname* must be preceded by a comma:

```
[, itemname] [...]
```

If a punctuation character precedes the ellipsis, you must use that character as a delimiter to separate repeated elements. However, if you select only one element, the delimiter is not required. In the following example, the comma should not precede the first instance of *itemname*:

```
[itemname] [, ...]
```

...

Within examples, horizontal ellipsis indicate where portions of the example are omitted.

INTRODUCTION

MPEX (MPE eXtended) is an extremely useful utility that improves the way you do your system management, program development and console operations tasks.

- It lets you EXECUTE MPE COMMANDS (like :COPY, :COBOL, :PRINT, :RENAME, etc.) on FILESETS. A single MPEX command can take the place of hundreds of MPE commands, saving you time and reducing the possibility of error.
- Commands like :ABORTJOB, :BREAKJOB, etc. are executed on USERSETS.
- It lets you SELECT FILES by various attributes like disk space used, last access date, by file owner, by disk number, etc. — something you just can't do without MPEX.
- It implements NEW COMMANDS like %ALTFILE that can do things no existing MPE command can do, such as alter a file limit, change file's location, etc. %EDIT command will execute several EDITOR commands on many files at once, and so on.
- With its MPEX HOOK feature, MPEX lets you execute REDO, :RUNS, UDCs, and MPEX COMMANDS from within virtually any utility, such as EDITOR, QUERY, etc. And more non-fileset-handling useful commands are available ([FILTER](#), [GOON](#), [DEVCONTROL](#), [ALARM](#), etc.) - see the manual for details.
- MPEX implements virtually all of the MPE/iX user interface features (UDCs, variables, command files, implied :RUN, :CALC, :COPY, :PRINT, etc.).

IMPORTANT: this feature lets you "live" in MPEX!

MPEX is not a replacement of or change to the existing operating system. It is a stand-alone program (just like EDITOR, QUERY, etc.).

To use MPEX (see also [Running MPEX](#) in this manual), simply type:

```
:RUN MAIN.PUB.VESOFT
```

You should see the MPEX product banner and version number, a hint, and then you will be prompted with MPEX's "%" prompt; you can then type in any MPEX command or MPE command (including :RUN, UDCs, etc.).

To get help on any command or operation, type

```
:HELP
```

To see our "self-demo", just type

```
%DEMO
```

and the fundamental MPEX features will be explained with working examples.

To exit MPEX, simply type "E", "END", "EXIT" or "QUIT".

Note: MPEX emulates the MPE/iX user interface. Because some of our users may not be familiar with MPE/iX, we define all our MPE/iX-compatible features from scratch without requiring any prior MPE/iX knowledge. However, even readers who are familiar with MPE/iX should still read those sections as we often go beyond the standard MPE/iX features.

INTRODUCTION: Examples of MPEX commands

POSIX As of MPE/iX version 5.0, POSIX compatibility has been added to the MPE operating system.
note: Where appropriate, POSIX notes such as this one will be included in the text, pointing out differences between normal MPE/MPEX commands and enhancements for POSIX.

Examples of MPEX commands

An example is worth a thousand syntax diagrams. Here are only a few of the things you can do with MPEX. We won't discuss each one in too great a detail — see the appropriate manual section for a complete explanation. (Remember also that these examples are only the tip of the iceberg; there are many more features discussed further in the manual.)

System management

```
%ALTFILE @.@(OWNER="TOM.VESOFT"); OWNER=MANAGER.VESOFT
```

After you purge the user `TOM.VESOFT`, this command will change all of his files to have owner ID `MANAGER.VESOFT`.

```
%ALTFILE @.DATA-CUST@.DATA(INTCODE=0); FLIMIT=EOF*1.5
```

Sets the file limit of the (MPE and KSAM) files in the `DATA` group (except for files that start with `CUST`) to be 50% greater than the current EOF.

```
%ALTSPoolFILE $STDLIST.@.(SPOOL.JSNAME="MCOMPILE"); PRI=8
```

Changes to 8 the output priority of all `$STDLIST`s belonging to jobs that logged on with job/session name `MCOMPILE`.

```
%COPY @.@.AP, @/=.@.APBACKUP; KEEPATTR
```

Copies all the files in the `AP` account into the `APBACKUP` account, preserving file attributes (e.g. creator ID, file limit, lockword, etc.).

```
%COPY @.DATA, =.=; YES; DEV=ROBIN#
```

Copies all the files in the group `DATA` (including `IMAGE` databases and `KSAM` files!) over a `DSL` (or `NS LAN`, etc.) to system `ROBIN`, purging any files that already exist with the same name.

```
%DELETESPOOLFILE $STDLIST.@.(SPOOL.OUTPRI<=2)
```

Deletes all the `$STDLIST`s that have output priority 2 or less.

```
%LISTF MYFILE, ACCESS
```

Shows all the accessors of `MYFILE`: accessing program name, access job number/name, current record being accessed, etc.

```
%LISTF @.DATA, DISCUSE
```

Shows which devices the files in the "DATA" group reside on (invaluable if you're interested in disk balancing).

```
%MPEXSTORE @.@.@ - @.@.SYS - @.@.VESOFT &
    (ACCDATE<TODAY-90 and NOT ISPRIV); *T; SHOW
%MPEXSTORE / - /SYS/ - @.@.VESOFT &
    (ACCDATE<TODAY-90 and NOT ISPRIV); *T; SHOW
```

Implements the sophisticated MPEX FILESETS (combining MPE and HFS syntax, selection criteria, etc.) on the MPE :STORE command.

```
%REPEAT
%>ECHO -----Formatting !MPEXCURRENTFILE
%>RUN MYFMT;INFO="!MPEXCURRENTFILE"
%>FORFILES X@.SOURCE
```

Performs a user-specified set of commands (here ECHO and RUN) on a fileset.

```
%SHOWOUT $STDLIST.@.@(SPOOL.OUTPRI=1)
```

Shows information on all \$STDLISTS with output priority 1 in the system, including job number, job name, ready date, etc.

```
%SHOWOUTJ COMP010,@.PROD
```

Shows information on all the spool files created by a job that logged on with job name COMP010 in the PROD account.

Managing disk space

```
%ALTFILE @.@.@(SAVABLESECTORS>0 and NOT OPENED);XLTRIM;KEEPAMDATES
```

On MPE/iX systems, this command releases a lot of currently unused disk space that has been allocated to files, without changing the capacity (FLIMIT) of the files in any way. (The ";KEEPAMDATES" option tells MPEX not to change the last access/modify dates of any files it alters, so they won't appear on your next partial backup.)

```
%COPY @.@.MFG, =.=.MFGBACK
```

Makes a backup copy of the MFG account to the MFGBACK account.

```
%LISTF @.DATA(ISASCII and SAVABLESECTORS>100),2
```

Lists all the ASCII files with part of the last extent unused. By XLTRIMing them one might save more than 2000 sectors on some files.

INTRODUCTION: Examples of MPEX commands

```
%ALTFILE RPT@.DATA(ISASCII); SQUEEZE
```

"Squeezes out" all the unused space between EOF and file limit in all blank-code files that start with RPT in the DATA group.

```
%EDIT @.SOURCE(CODE="EDTCT"), SET VARIABLE
```

Converts all EDITOR COBOL-format files in the SOURCE group to be variable record length files, saving 30-50% of the disk space while still leaving them editable, compilable, etc.

```
%LISTF @.@.(ACCDATE<TODAY-120), 3
```

Finds all the files that haven't been accessed in the last 120 days and lists their names, access dates, modify dates, etc.; also shows total number of files and sectors.

```
%PURGE K#####.@.+LOG####.@.(CREDATE<TODAY-7)
```

Purges EDITOR K-files and system log files created more than a week ago. Prompts you for confirmation before purging anything!

```
%PURGE @.@ - @.@.VESOFT(ACCDATE<TODAY-120 and NOT ISPRIV)
%PURGE / - @.@.VESOFT(ACCDATE<TODAY-120 and NOT ISPRIV)
```

Purges all the non-privileged (mostly non-IMAGE) files that haven't been accessed in the last 120 days.

Program development

```
%COBOL AP@.SOURCE,AP@.PUB,*LP
```

Compiles using COBOL and :PREPs all the AP@.SOURCE files into program files with the same name in the PUB group. (Of course, the same works for FORTRAN, PASCAL, RPG, etc.

```
%PRINT @.SOURCE;SEARCH="ITEM-NUM" or "ITEM-NAME" or "ITEM-DESC";NUM
```

Finds all the occurrences of the strings "ITEM-NUM" or "ITEM-NAME" or "ITEM-DESC" in all files in the SOURCE group.

```
%PRINT @.SOURCE;SEARCH=DELIM "MYVAR";PREV="PROCEDURE" or "FUNCTION"
```

Prints all occurrences of the string "MYVAR" — for each occurrence, also prints the last prior occurrence of the line that contains the strings "PROCEDURE" or "FUNCTION"; in other words, shows you all the lines that refer to "MYVAR" AND indicates what procedure they occur in!

DELIM means that "MYVAR" will be searched for as a delimited string, i.e. surrounded by special characters (e.g. lines containing "TEMPMYVAR" or "MYVARIATION" will not be found).

```
%COBOL @.PRODSRC(FSEARCHEXP("'CUST-NO' or 'CUST-NAME'")>0), =.PUB
```

Recompiles all the files in PRODSRC that contain within them either the string CUST-NO or the string CUST-NAME!

```
%EDITCHG @.SOURCE, "ITEM-NO", "ITEM-NUMBER"
```

Changes all occurrences of "ITEM-NO" to "ITEM-NUMBER" in all files in the SOURCE group, /KEEPing only those files that were actually changed.

```
%LISTF @.PRODSRC(MODDATE>3/15/1991), 3
```

Finds all the files in PRODSRC that have been modified since 3/15/1991 (perhaps the last production release date).

```
%PRINT @.SOURCE; OUT=*LP; PAGEHEAD; NUM
```

Prints all the SOURCE group files to the line printer (with page headers, nicely-formatted line numbers, etc.).

```
%COBOL @.PRODSRC(MODDATETIME>VEFINFO(FILE+".PUB").MODDATETIME), =.PUB
```

Compiles all PRODSRC files whose modify dates and times are later than the modify dates and times of files with the same name in the PUB group (VEFINFO(FILE+".PUB").MODDATETIME). In other words, this recompiles all the programs that have been modified but not recompiled into PUB.

Fancy, eh?

Database management

```
%ALTFILE CUST?#.DB(DBSETTYPE="D"); DEV=2
%ALTFILE CUST?#.DB(DBSETTYPE<>"D"); DEV=3
```

These two commands move all detail datasets of the CUST database to device 2 and all non-detail datasets to device 3.

```
%ALTFILE @.@.AP(CREATOR<>"DBA" and ISPRIV); CREATOR=DBA
```

Changes the creator ID of all AP databases (ISPRIV) to be DBA, thus making sure that DBA — and only DBA — can run DBUTIL, DBSTORE, etc. against all those databases.

```
%COPY APDB@, DBNEW@; DEV=3
```

Copies the APDB database into a new database called DBNEW, moving all the files onto disk device 3.

INTRODUCTION: Examples of MPEX commands

```
%DBADGALT @.DB(DBSETFULLNESS>.8 and MODDATE>TODAY-30), 0.6  
%DBGENALT @.DB(DBSETFULLNESS>.8 and MODDATE>TODAY-30), 0.6
```

Use ADAGER, DBGGENERAL/BRADMARK, etc. to alter the capacity of all datasets in the DB group that are currently more than 80% full (and have been modified within the last 30 days) so that they will only be 60% full.

```
%LISTF @.@.AP(DBSETFULLNESS>.70 and MODDATE>TODAY-7), DB
```

Shows database information (dataset name, type, capacity, etc.) for all datasets in the AP account that are more than 70% full and have been modified in the past week.

```
%RENAME MYDB@(ISPRIV), NEWDB@
```

Renames a database.

File system security

```
%LISTF @.@.(PROG.PMCAP), 5
```

Finds all the privileged-mode program files in the system and lists their program file information.

```
%LISTF @.@.PAYROLL(ACCDATE>=TODAY-2)
```

On Monday, this shows all files in the PAYROLL account that were accessed over the weekend.

```
%SECURE @.@.(ISPROG and ISRELEASED and DIRGROUP.CAP("PM"))
```

Secures all the :RELEASEd program files in privileged groups (which are a SERIOUS THREAT to system security).

```
%LISTF @,SEC
```

Shows the "true" access to the file, factoring in both the group- and account-level restrictions as well as the file restrictions and the :RELEASE/:SECURE status.

Job stream programming

```
!JOB JOBA,MGR.PROD  
!RUN MAIN.PUB.VESOFT;PARM=1  
%WHILE JSCOUNT("JOB,MGR.PROD&BATCH")>0 DO  
% PAUSE 120  
%ENDWHILE  
%EXIT  
...
```

If the job JOBB,MGR.PROD is logged on, JOBA waits until JOBB logs off before continuing execution.

```

!JOB  JOBA,MGR.PROD
...
!RUN  MAIN.PUB.VESOFT;PARM=1
%WHILE VEFINFO("DATAFILE").OPENED DO
%  PAUSE 120
%ENDWHILE
%EXIT
...

```

If the file DATAFILE is opened, JOBA waits until it is closed before continuing execution.

```

!JOB  JOBA,MGR.PROD
...
!RUN  MAIN.PUB.VESOFT;PARM=1
%SETVAR FNAME PRINTOPREPLY('Which file should we output to?')
%IF FEXISTS(FNAME) THEN
%  IF PRINTOPREPLY('OK to purge file '+FNAME+'?')='Y' THEN
%    PURGE !FNAME
%  ENDIF
%ENDIF
%FILE MYOUTFIL=!FNAME,NEW;SAVE
%EXIT
...

```

JOBA asks the console operator for feedback via the :REPLY command and uses it to set up a file equation and possibly purge the file (if it already exists).

RUNNING MPEX

There are several ways in which you can run MPEX:

- Interactively, by saying

```
:RUN MAIN.PUB.VESOFT
```

- In BATCH, by saying

```
!JOB MYJOB,USER.ACCOUNT  
!RUN MAIN.PUB.VESOFT;PARM=1  
%mpex commands...
```

- With an ;INFO= string containing a single MPEX command that is to be executed:

```
:RUN MAIN.PUB.VESOFT;INFO="LISTF @.@.AP,DB;*LP"
```

- Using the "MPEX HOOK" from within a program such as EDITOR, QUERY, etc.:

```
:RUN EDITOR.PUB.VESOFT;LIB=P  
/TEXT MYFILE  
...  
/%COBOL MYFILE,MYPROG  
<< MPEX executes the %-prefixed command >>
```

All of these ways of running MPEX are subject to the following features:

- You may specify commands that are to be executed every time a user enters MPEX. This may be done on the system level, the account level, the group level or a combination of all three.
- You may (if your system manager allows this) tell MPEX not to recognize UDCs — this will substantially speed up entry into MPEX.

Running MPEX interactively

If you just say

```
:RUN MAIN.PUB.VESOFT
```

you'll enter MPEX's interactive mode (MPEX will prompt you with a "%").

In fact, many of our users like to "live" in MPEX — they just run MPEX when they log on (they might even have an `OPTION LOGON` UDC to do this) and don't get out until they're ready to do `%BYE`. This can be very useful because:

- Virtually every MPE command can be executed from within MPEX (so you almost never need to exit into the Command Interpreter).
- MPEX's command history feature (`%REDO`, `%LISTREDO` and `%DO`) is so useful that you might want to always have it available. With the MPEX HOOK, we make our powerful `%REDO` available in virtually every program (EDITOR, QUERY, DBUTIL, etc.).

- MPEX is a very powerful process-handling environment, with which you can manage multiple suspended processes, quickly pop into and out of programs, and so on.

(Alternatively, you might get all these features by "living" in the HOOKed EDITOR [or QUERY or whatever], in which you'll also have the command history and the full power of MPEX commands.)

Running MPEX in batch

If you `:RUN MAIN.PUB.VESOFT` in batch, MPEX will (just like MPE) abort after the first error (unless you do a `%CONTINUE` before the command).

What constitutes an error?

- Any MPE command that gets a CIERROR.
- Any program run that aborts (or actually any time a program terminates with the JCW set to FATAL).
- A fatal error on an MPEX command.

Not all MPEX commands that get an error cause a job stream to abort. What if you do a fileset `%COBOL` compile and one of a hundred compiles fails? Is that an error because one compile failed or a successful result because 99 compiles completed OK?

At the moment we don't treat this as an error, but we're considering changing it (if we decided that we need to for greater MPE compatibility). Keep this in mind — if you want the job to continue even though an error might occur in a particular command, always put a `CONTINUE` in front of the command JUST IN CASE.

What if you always want the job to continue, even if an error occurs in any of the commands?

If you say

```
%SETVAR HPAUTOCONTTOPTLEVEL TRUE
```

then MPEX will enter "auto-`:CONTINUE`" mode, in which command errors will NEVER cause the entire MPEX run to abort. To change back to default mode, say

```
%SETVAR HPAUTOCONTTOPTLEVEL FALSE
```

Note that if you run MPEX in batch and want to give MPEX commands that start with "!", e.g.

```
!JOB...
!RUN MAIN.PUB.VESOFT;PARM=1
%!LISTF @.@.(ACCDATE<TODAY-120),3
...
```

you must prefix the MPEX command with either a "%" (as we do here) or with a space — if the "!" is put in the first column, MPE automatically replaces it with a ":" behind our backs.

Commands that you don't prefix with a "!" need not have a leading space or "%"; however, it's probably a good idea to always prefix MPEX commands in your job streams with a "%" — it'll make it clear to anybody who's reading the stream that this is an MPEX command.

RUNNING MPEX: Running MPEX with ;INFO=

When you say

```
:ABC &  
:DEF GHI
```

in a job stream, MPE executes the command as "ABC DEF GHI" — even though the job stream does contain a leading ":" at the start of the "DEF GHI" line (or perhaps a leading "!"), it gets stripped out when the two lines are merged together.

If you said

```
:ABC &  
:DEF GHI
```

in older versions of MPEX, MPEX would NOT strip out the colon; instead, it would execute the command as ":ABC :DEF GHI".

For the sake of compatibility with these older versions, MPEX leaves the continuation line's colon in; however, if you set the JCW MPEXCONTINUECOLON to 1, MPEX will instead do the same thing as MPE does (i.e. strip out the leading colon in the continuation line). We wish that we could make this the normal behavior, but we feel that we need to remain compatible with older versions of MPEX, especially in situations which might happen in production job streams.

When you run `CI.PUB.VESOFT`, the leading colons in continuation lines will always be stripped out (just like in MPE). This is because `CI.PUB.VESOFT` is intended to be as compatible as possible with MPE.

Running MPEX with ;INFO=

If you say

```
:RUN MAIN.PUB.VESOFT;INFO="command"
```

MPEX will execute the command and return to MPE. This is the "immediate" mode, used when you want to execute just one specific MPEX command (often from a UDC or a command file).

The only special things you need to know about immediate mode are:

- MPEX won't output the header before executing the command.
- If you want to speed up this operation (perhaps by turning off UDCs initialization, which may not be needed if you know exactly what command you're executing), see the [Speeding Up MPEX Start-up](#) section below.

Specifying commands to be executed each time MPEX is entered

MPE's `OPTION LOGON` UDCs let you define commands to be executed every time you log on — MPEX also has a mechanism for specifying commands to be executed every time MPEX is entered.

When MPEX is run, the first thing it does is execute the command file `MPEXMGR.PUB.VESOFT`.

For instance, if `MPEXMGR.PUB.VESOFT` contained the lines

```
INITUDCS
SET  CAPABILITY,PURGE,AM
SET  DEFAULT,PURGE,?
```

then these lines would be executed every time any user enters MPEX. UDCs would be initialized (`INITUDCS`), the MPEX prompt would be changed (`SETVAR MPEXPROMPT`), and the `PURGE` command would be additionally secured (`SET CAPABILITY` and `SET DEFAULT`).

The `MPEXMGR.PUB.VESOFT` file might contain, for example:

```
UNLESSFAST CALC READHINT('MPEXHINT.HELP.VESOFT')
INITUDCS
UNLESSFAST INITMPEXMGR
INITREDO
```

What does this mean?

- The `UNLESSFAST CALC READHINT('MPEXHINT.HELP.VESOFT')` prints a random "hint" — one- or two-line message about a feature of MPEX that you may or may not be familiar with (this is done essentially for educational purposes). The "UNLESSFAST" means that the hint should not be printed if you're using the fast MPEX start-up option (see below).
- The `INITUDCS` command makes MPE UDCs accessible from MPEX.
- "`UNLESSFAST INITMPEXMGR`" says that MPEX should execute the "`INITMPEXMGR`" command unless the fast MPEX start-up option (which will be explained shortly) has been selected.

An `INITMPEXMGR` command tells MPEX to execute the contents of the `MPEXMGR.PUB` file in your logon account and the `MPEXMGR` file in your logon group. If one or both of those files doesn't exist, no problem; however, if they do, they'll be executed just like the `MPEXMGR.PUB.VESOFT` file is.

Note that commands you put before the `INITMPEXMGR` command can be "overridden" by commands in a user's own `MPEXMGR` file(s). The exception is "`SET CAPABILITY`" commands, which CANNOT be removed by users at all (see "`%SET CAPABILITY`" in this manual). For similar reasons, you should place any commands that MUST be executed prior to the `INITMPEXMGR` command. If a user's own `MPEXMGR` file contains an error, processing of the `MPEXMGR` file(s) will stop at that point.

Again, the reason that MPEX doesn't automatically execute the `MPEXMGR` and `MPEXMGR.PUB` files is for speed of start-up — by having `MPEXMGR` and `MPEXMGR.PUB` execution controlled from the `MPEXMGR.PUB.VESOFT` file, we let you omit it entirely or put an "UNLESSFAST" in front of it.

- `INITREDO` is not documented elsewhere, since it is only really useful in the `MPEXMGR.PUB.VESOFT` file. `INITREDO` simply initializes MPEX's `%REDO`, `%DO` and `%LISTREDO` command handling structures.

The only reason that MPEX doesn't automatically do this (but rather requires an `MPEXMGR` command to do it) is that this may take some time, and we want to allow you to make this step (`REDO` initialization) optional. See "[Speeding up MPEX start-up](#)" below.

RUNNING MPEX: Specifying commands to be executed each time MPEX is entered

Other useful commands you might want to include into the `MPEXMGR.PUB.VESOFT` file:

- `%SET CAPABILITY` commands to restrict access to certain MPEX commands, e.g.

```
%SET CAPABILITY,PURGE,AM
```

- `%SET DEFAULT` commands to change the default execution mode for MPEX commands, e.g.

```
%SET DEFAULT,PURGE,?
```

to make sure that (by default) all purges will ask for yes/no verification.

- `%SET DATE` commands to change the input format for MPEX dates; our European customers like to say

```
%SET DATE,DMY
```

- Various `:SETJCW` commands to control the behavior of MPEX's, like `REDO` command history (see "[%REDO Facility Options](#)" in the `%REDO` chapter).

Your `MPEXMGR.PUB` and `MPEXMGR.logongroup` files are also executed at MPEX entry time, but after the `MPEXMGR.PUB.VESOFT` file. (Note: you do not need Read access to the `MPEXMGR` files, only execute). Therefore:

- If your `MPEXMGR.PUB.VESOFT` file already has an `%INITUDCS` command and an `%INITREDO` command, there's no reason to put them into your own `MPEXMGR` files — the UDCs and the redo facility are already initialized; reinitializing them would just waste time. (In fact, under certain circumstances, it might actually give you some unpleasant error messages!)
- Never put an `%INITMPEXMGR` command into your `MPEXMGR.PUB` or `MPEXMGR.logongroup` file. "`%INITMPEXMGR`" is essentially the same as "`%XEQ MPEXMGR.PUB`" and "`%XEQ MPEXMGR.logongroup`"; if you put it into one of those files, it'll loop and eventually cause a stack overflow.
- In general, do not copy the default `MPEXMGR.PUB.VESOFT` into your own group or `PUB` group, since the commands that `MPEXMGR.PUB.VESOFT` usually contains — `%INITMPEXMGR`, `%INITREDO` and `%INITUDCS` — are not appropriate in your own `MPEXMGR` files.

Other commands, though — `%SETs`, `%SETVARs`, etc. — can certainly be put into your `MPEXMGR.PUB` and `MPEXMGR.logongroup` files. Remember that, in general, `MPEXMGR.PUB.VESOFT`, `MPEXMGR.PUB` and `MPEXMGR.logongroup` can contain any MPEX commands.

Important security considerations for the MPEXMGR file

As noted above, the `MPEXMGR` file is a great place to put `%SET CAPABILITY` commands. However, anyone who has `WRITE` access to an "`MPEXMGR`" file could conceivably change it to do something other than intended, such as removing those `%SET CAPABILITY` commands that you put in the file. While the `MPEXMGR.PUB.VESOFT` file should remain secure if the group and account attributes we set up during installation are not changed, your account level `MPEXMGR` files might not be so safe.

For similar reasons, you should place any critical commands `PRIOR` to any `INITMPEXMGR` command. Since `INITMPEXMGR` will execute the files `MPEXMGR.PUB` and `MPEXMGR` in the user's account and group respectively, any commands that a user places in these files may be executed `BEFORE` the appropriate `%SET CAPABILITY` command(s) in `MPEXMGR.PUB.VESOFT`.

Also, careless or intentional mistakes in these files can stop the entire MPEXMGR start-up sequence, leaving the system unprotected from the user.

One last thing to be aware of, although not specifically related to security, is that any commands that appear AFTER the INITMPEXMGR command in MPEXMGR.PUB.VESOFT may override commands and settings in the user's own MPEXMGR file. Sometimes this is desired, but often it is not. So if some items should have a default, but individual users may be allowed to change that default (such as the HPPATH or MPEXPROMPT variables), defaults should be placed BEFORE the INITMPEXMGR command.

Speeding up MPEX start-up

Like so many good things, some of what MPEX does at start-up — before you type in your first command — takes time. As a rule, MPEX must:

- Figure out what MPE UDCs you have set up; this includes:
 - FOPENing COMMAND.PUB.SYS;
 - Finding three directory entries to determine what UDC files you have set;
 - Reading several COMMAND.PUB.SYS records to figure out all your UDC filenames;
 - FCLOSEing COMMAND.PUB.SYS;
 - FOPENing every one of those UDC files;
 - Reading each UDC file to determine what UDC commands it contains (and remember their names and record numbers).

(The CI has to do all the above when you log on — this is why logons can sometimes take a while. We have to do this again since the CI's UDC tables are inaccessible to us.)

- Open your REDO command history file.
- Open your MPEXMGR.PUB and MPEXMGR files, and execute all the commands kept in each of them. (Even if you don't have these files, the FOPEN attempts take time.)
- And many other things that can't be avoided in any case.

The MPEXMGR.PUB.VESOFT command file — which is executed whenever you enter MPEX — lets you control which of the above operations are done and which are not.

What's more, it lets you have two modes of operation (which you can select at :RUN MAIN.PUB.VESOFT time):

- A normal mode, which gives you the full power of UDCs, the REDO command history and MPEXMGR files.
- A fast-start mode, which doesn't support some or all of the above, but is faster at start-up time. To enter it, you say

```
:RUN MAIN.PUB.VESOFT;PARM=1
```

The ;PARM=1 indicates fast-start mode.

If you have no control over how MPEX.PUB.VESOFT will be run (e.g. if you're using the MPEX HOOK facility, which runs MPEX for you), you can trigger fast-start mode by saying

```
:SETJCW MPEXFASTSTART=1
```

RUNNING MPEX: Speeding up MPEX start-up

before MPEX is run. (Remember that this will apply to all subsequent runs of MPEX in your session until you say `:SETJCW MPEXFASTSTART=0`.)

For instance, say that your `MPEXMGR.PUB.VESOFT` file says the following:

```
UNLESSFAST :CALC READHINT('MPEXHINT.HELP.VESOFT')
UNLESSFAST :INITUDCS
UNLESSFAST INITMPEXMGR
:INITREDO
```

What does this mean?

- The random "hint" — one- or two-line description of some MPEX feature that we want to make sure you know about — is output unless you are in fast-start mode.
- UDCs are to be initialized unless you are in fast-start mode. UDC initialization is the most time-consuming of all the start-up tasks.
- `MPEXMGR.PUB` and `MPEXMGR.yourgroup` files are to be executed unless you are in fast-start mode.
- `REDO` command history will always be initialized (even in fast-start mode). We recommend this since initialization is so quick (just one `FOPEN`) and the command history is so useful.

(The ":"s were also put in front of `INITREDO` and `INITUDCS` [which are not MPE commands] for speed reasons — using them prevents MPEX from trying to interpret them as "true" MPEX commands, which would take a couple more disk I/Os. It's complicated — just trust us.)

The default `MPEXMGR.PUB.VESOFT` file that we give you says

```
IF HPINTERACTIVE THEN
  INITREDO
  CONTINUE
  NOMSG UNLESSFAST :CALC READHINT('MPEXHINT.HELP.VESOFT')
ENDIF
INITUDCS
UNLESSFAST INITMPEXMGR
```

We don't put an "UNLESSFAST" before the `:INITUDCS` for security reasons — some system managers use UDCs to block out MPE commands (such as `:SHOWCATALOG`), and we don't want to give everybody an easy way (running MPEX in fast-start mode) to avoid this restriction.

If, however, you have no security problems with allowing people to avoid UDCs, we suggest that you put an `UNLESSFAST` in front of the `:INITUDCS`. This will make it easy for people to get into MPEX very quickly (if they need to).

Let us point out, though, that the time it takes to get into MPEX isn't so long at all, and the UDC support, `REDO` command history and `MPEXMGR` features are actually quite valuable.

We envision people using the fast-start option only selectively when they want to do one thing with MPEX and want to do it quickly. For instance, if you have a UDC from which you want to run MPEX with an `;INFO=` string to execute a specific MPEX command, you might want to use `;PARAM=1` because you know that you won't need UDCs, `REDO`, or local `MPEXMGR` files.

CI.PUB.VESOFT (for MPEX/iX CI emulation)

In MPE/iX, the command interpreter is implemented as a separate program named `CI.PUB.SYS`. You can, for instance, say

```
:CI.PUB.SYS
```

and create a son CI process beneath your normal CI. `:EXIT` returns you to the father CI. A typical use for this is when you want to run the CI with `STDIN` or `STDLIST` redirected:

```
:FILE MYFILE,NEW;SAVE
:RUN CI.PUB.SYS;INFO="SHOWOUT";PARAM=3;STDLIST=*MYFILE
```

will make the CI do a `SHOWOUT` with output to the `;STDLIST=` file. The `;INFO=` string is the command to be executed; the `;PARAM=` contains some flag settings we'll discuss shortly.

MPEX allows you to do something quite similar. At installation time, the MPEX program file is copied into the file `CI.PUB.VESOFT` — when you run this program, MPEX will try to emulate (even more closely than it normally does) the behavior of MPE/iX's `CI.PUB.SYS`.

In particular:

- The prompt will default to a `":` (rather than a `"%`"); to change the prompt, you should set the variable `HPPROMPT` (not `MPEXPROMPT`).
- Commands entered into `CI.PUB.VESOFT` will be executed as MPE or MPE/iX commands, not as MPEX fileset handling commands. (This is the same behavior as if you had `RUN MAIN.PUB.VESOFT` and prefixed each command you typed with a `":`.) MPE/iX commands (like `:COPY`, `:PRINT`, `:INPUT`, etc.) will still be available, but without fileset handling.
- Instead of looking at the `MPEXCMDTRACE` variable (to see if command tracing should be on), `CI.PUB.VESOFT` looks at the `HCMDTRACE` variable.
- Instead of executing `MPEXMGR.PUB.VESOFT`, `CI.PUB.VESOFT` executes `CIMGR.PUB.VESOFT`. If `CIMGR.PUB.VESOFT` has an `INITMPEXMGR` command, it executes `CIMGR.PUB.logonaccount` and `CIMGR.logongroup.logonaccount`.

If `CI.PUB.VESOFT` is run with an `;INFO=` parameter, the `;INFO` string will be executed.

Precise behavior depends on whether `;PARAM=` is also specified, and the value supplied:

<code>;PARAM=</code>	Display CI Header?	EXIT after executing <code>;INFO=</code> string?
0 (default)	Yes	No
1	Yes	Yes
2	No	No
3	No	Yes

`CI.PUB.VESOFT` will emulate the behavior of MPE/iX even more closely than MPEX normally does; however, MPEX fileset handling commands will not be available.

USING MPEX

Changing the MPEX prompt

One of the things that MPEX lets you do is change the MPEX prompt from its default value of "%". By setting the MPEX variable `MPEXPROMPT` you can set the prompt to whatever you please. For instance, you might say

```
%SETVAR MPEXPROMPT "!!HPCMDNUM/!!HPUSER/!!HPTIMEF %"
```

then MPEX will prompt you with something like:

```
182/EUGENE/10:15 AM %
```

Now, whenever you look at the prompt, you'll see the current command number (useful for future `%REDOs`), your user name (good for reminding you which computer or logon you are using if you connect to multiple CPUs), and the current time of day.

As you see, the `MPEXPROMPT` may include MPEX variables that are substituted for when the prompt is printed. `HPCMDNUM` and `HPTIMEF` are common examples of what people might put into their prompts, but you can actually include any variable or constant you want.

Note that we use two "!" characters in this particular `%SETVAR` command. This is because when MPEX sees the "!" character, it tries to evaluate the variable before executing the command, so your `MPEXPROMPT` will not be dynamically evaluated. Seeing two "!" characters in the command, MPEX translates this into a single "!" character in your prompt variable which causes MPEX to evaluate it AGAIN when displaying your prompt. See the [A few comments about the "!" character](#) section of the MPEX VARIABLES chapter.

Running under MPEX/iX

MPE/iX (formerly known as MPE/XL) is the version of the MPE operating system created for the HP3000 series 9xx (aka SPECTRUM, RISC, HPPA or PA-RISC) systems. While the MPE/iX Command Interpreter (the "CI") provides a number of enhancements to the old MPE/V CI, it still does not address any of the fundamental improvements MPEX has made available to HP3000 users for many years, helping them save time and improving the reliability of their work.

MPEX eXtends MPE/iX in several ways:

- MPEX lets you execute MPE commands — like `:COPY`, `:COB85XLK`, `:RENAME`, `:PRINT`, etc. — on entire filesets (including temporary filesets, IMAGE db filesets, and — for commands like `:ALTSP00LFILE`, `:DELETESPOOLFILE`, etc. — on spool filesets)!
- MPEX also lets you execute MPE job-handling commands — like `:ABORTJOB`, `:BREAKJOB`, `:RESUMEJOB`, etc. — on entire usersets.
- MPEX's selection conditions allow you to:
 - select files by hundreds of different attributes such as "disk space used", "last access date", "file creator", etc.;

- select program files by attributes like "has PM capability", "number of code segments", "is OCTCOMPed", etc.;
- select database files by attributes like "dbset name", "dbset fullness", "dbset block wastage", etc.;
- select spoolfiles by attributes like "state (active, ready, opened, locked)", "job number", "job aborted", etc.;
- select jobs by attributes like "jobname", "intro time", "state (EXEC, SUSP, WAIT, INIT, SCHED)", etc.
- MPEX implements entirely new commands that do things no existing MPE/iX command does; for example:
 - %ALTFILE (which lets you change attributes of files, and gives you the very useful ;XLTRIM disk-space-saving option);
 - %CHLOGON (which lets you switch to another logon without doing a :HELLO);
 - %PAUSEJOB (which suspends jobs until a future time or condition).
- It also implements new control structures like
 - %REPEAT...%FORFILES (which lets you execute arbitrary commands against MPEX filesets).
- MPEX HOOK gives you:
 - a REDO facility that lets you redo up to the last 1000 (or more) commands that you've typed into EDITOR, TDP — virtually any program;
 - the ability to execute any MPEX command directly within HOOKed utilities.
- MPEX greatly enhances the MPE/iX user interface by providing:
 - new functions and variables, including hundreds of file attribute variables and functions; integer, boolean, string, real, date and time functions; file and database I/O functions; special MPE access functions; terminal manipulation functions; and security-related functions (see [Appendix B](#) for details); and
 - enhancements to existing commands, like %PRINT...;SEARCH=... (to search for text within filesets), new %LISTF modes, and %COPY (which allows you to copy IMAGE databases, copy files across NS, etc.).

Entering MPE commands in MPEX (':' and 'MPE' prefixes)

Syntax:

```
%[:] command or UDC [command parameter(s)...]  
%MPE command or UDC [command parameter(s)...]
```

Examples:

```
%PURGE @.DATA(ACCDATE<TODAY-120 and NOT ISPRIV)  
%:FILE X;REC=-80,,F,ASCII;NOCCTL;SAVE  
%MYPROG;PARM=123;MAXDATA=30000;LIB=G  
%S J << where S is a valid UDC >>  
%MPE LISTF MYFILE,3
```

MPE commands can be executed from within MPEX by just typing them. Even though most MPEX commands have the same name as their corresponding MPE command, they are still

Using MPEX: YES/NO prompting ('?' prefix)

being interpreted and executed by MPEX; you can speed up some of these commands by prefixing them with a ":" or the word "MPE".

Also, most MPEX commands are upwardly-compatible with MPE in that MPEX implements some EXTRA features while supporting the normal MPE syntax; however, some commands are not upwardly-compatible with MPE — for instance, the second parameter in the %COBOL command is the program filename, not the USL file name.

(This way seemed more convenient, but if we had to do it over again, we'd have called the command %COBOLPREP to make everything perfectly compatible.) If you run into one of these incompatible commands and want to execute the MPE version of it rather than the MPEX version, you can either type a leading ":" like this:

```
%:COBOL MYSOURCE,MYUSL
```

or preface the entire command with "MPE" like this:

```
%MPE COBOL MYSOURCE,MYUSL
```

The difference between these two forms is that the colon prefix indicates to MPEX that the command does not contain a fileset (you will get an error if you supply one), and the "MPE" prefix causes MPEX to pass the command directly to the COMMAND intrinsic.

For example, prefacing a %PRINT command with a ":" will speed up the execution of the command (since MPEX does not have to do any fileset processing), while still allowing the use of our extensions to the :PRINT command (such as ;SEARCH=... or ;FORMAT=...). However, prefacing a %PRINT command with "MPE" will generate an error if you try to use any extended feature of the %PRINT command.

YES/NO prompting ('?' prefix)

Syntax:

```
%%? command
```

Examples:

```
%%?PURGE @.DATA(ACCDATE<TODAY-120 and NOT ISPRIV)
OK to process ABACUS.DATA (y/n)? Y
-----Purging ABACUS.DATA
OK to process ABIGAIL.DATA (y/n)? N
OK to process ABRACADA.DATA (y/n)?      << [Return] is pressed >>
OK to process BANANA.DATA (y/n)? Y
-----Purging BANANA.DATA
...
```

If you prefix an MPEX fileset-handling command with a "?", MPEX will prompt you with

```
OK to process filename (y/n)?
```

If you type N or press [RETURN], the file will be skipped. Only if you type Y will the command be executed against the file.

This can be very useful if you know you want to do something to a somewhat ill-defined set of files (not something that can be exactly specified with +filesets, - filesets and selection

conditions). You could specify a fileset that contains all the files you want to work on, and then just type N when prompted for any of the files you don't want.

Naturally, this can also be particularly good for commands such as %PURGE, in which you want to be doubly sure before going ahead with the operation. This is why many of our users add to their MPEXMGR.PUB.VESOFT file (see "[Specifying commands to be executed each time MPEX is entered](#)" in the Running MPEX section of this manual).

```
%SET DEFAULT,PURGE,?
```

This means that whenever somebody types

```
%PURGE...
```

MPEX will act as if he had typed

```
%?PURGE...
```

If he doesn't want yes/no prompting, he'll have to explicitly specify this by saying

```
%!PURGE...
```

The "!" says "execute this in standard online mode", as opposed to "?" (with yes/no verification) or "\$" (in batch — see below).

Batch execution ('\$' prefix)

Syntax:

```
$$command
```

Examples:

```
$$COBOL AP@.SOURCE,AP@.PUB
$$DELETESPOOLFILE $STDLIST.@.(SPOOL.OUTPUT<=3)
$$RUN MYPROG;LIB=G;PARAM=100
```

Whenever you prefix a command (an MPE command or an MPEX command) with a "\$", that command will be executed OFFLINE, in a specially created job stream with the following characteristics:

- It will log on with a job name equal to the first 8 characters of the command you typed. This can be overridden by setting the variable MPEXDEFAULTJOBNAME to a specific job name.
- The job stream will log on with the same user, account and group names that you logged on with.
- It will log on with job parameters ";OUTCLASS=,1" — thus the \$STDLIST will go to the spooler but will not print since it will be deferred. You can alter this using the SUBMIT command or by setting the variable MPEXDEFAULTJOBPARMS to the desired default job parameters, e.g.

```
%SETVAR MPEXDEFAULTJOBPARMS "OUTCLASS=,2;PRI=ES"
```

Using MPEX: Batch execution ('\$' prefix)

For instance, if you're logged on as "JANE.AP,DEV" and you execute the %\$COBOL command shown above, the job will log on as "COBOL,JANE.AP,DEV;OUTCLASS=,1" if no overriding variables have been set.

- When MPEX builds the job stream, it will copy into it all the file equations that are then set in the session — thus, if you say

```
%FILE LISTFILE;DEV=LP
%FILE COPYLIB=MYCOPY.SOURCE
%$COBOL AP###S,AP###P,*LISTFILE
```

then the generated job stream will inherit the file equations for LISTFILE and COPYLIB (as well as any other file equations you might have in your session).

- MPEX will also copy your current HPPATH variable to the job.
- With the %\$SCHEDULE and %\$SUBMIT commands, you can also submit jobs that:
 - log on as some other user ID,
 - with some other job name,
 - with some other :JOB card parameters,
 - at some other time (i.e. with scheduling parameters).

Normally, whenever you do a \$command, a %\$SUBMIT or a %\$SCHEDULE, MPEX will prompt you with "OK to stream (Y/N)?" to verify that you really want to submit the job. If you don't want this, just say

```
:SETJCW MPEXSTREAMNOVERIFY=1
```

before you enter your "\$", "%\$SUBMIT" or "%\$SCHEDULE" command — MPEX will then submit the jobs without prompting for verification.

If you have run GOD.PUB.VESOFT (documented elsewhere in this manual) in your session and you do a

```
:SETJCW MPEXSTREAMGOD=1
```

before you enter your "\$", "%\$SUBMIT" or "%\$SCHEDULE" command, the job stream created by MPEX will automatically run GOD.PUB.VESOFT as well. If, however, you have not run GOD in your session or you haven't set MPEXSTREAMGOD to 1, GOD will not be run in the job stream.

After the job has completed, it will attempt to do a :TELL to your session to let you know either that the job is done or that an error occurred.

The message will include, if possible:

- the name of the system (if the variable HPSYSNAME is set — useful if you have multiple REMOTE sessions and jobs running);
- the job name (usually the first 8 characters of the command you typed — see [above](#));
- the output spool file number of the job's \$STDLIST (this lets you read the \$STDLIST without having to do a %\$SHOWOUTJ to find it);
- the text of the error message (if there was an error); and the name of the file on which the error occurred (if appropriate).

Normally, this `:TELL` is done to the job/session number that issued the `$`, `%SUBMIT` or `$SCHEDULE` command, but you can change this using the variable `MPEXTELLBACK`. Setting `MPEXTELLBACK` to `"!!HPJOBNAME,!!HPUSER.!!HPACCOUNT"` makes MPEX jobs send the message to all sessions logged on with your job/session name, your user ID and your account name. Setting it to `"#S!!HPJOBNUM"` (the default) makes MPEX jobs send the message to your session specifically (if you log off and log back on, no message will arrive). You can set this variable to anything you please — e.g. you might set it to `"MANAGER.SYS"` to make all jobs send messages to `MANAGER.SYS`.

If you like, you can replace the `:TELL` command in the job with either a `:TELL` or MPEX's `%WARNF` command by setting the `MPEXTELLBACKCMD` variable. For example:

```
%SETVAR MPEXTELLBACKCMD "WARNF"
```

(Note, of course, that you must be `:ALLOW`d the `:WARN` command in order to use `%WARNF`).

You can use MPEX's `%PRINTO` to view the `$STDLIST` of the last job you submitted via `"$"`, `"%SUBMIT"`, `"%SCHEDULE"` or using SECURITY's `STREAMX` (see the SECURITY User Manual for details on `STREAMX`) without having to specify the job number or spool file number! `%PRINTO` is documented in the MPEX Commands section of this manual.

Sometimes, you might decide that you'd normally want all, say, `%COBOL` commands or `%EDIT` commands to be done off-line. If you say

```
%SET DEFAULT,COBOL,$
%SET DEFAULT,EDIT,$
```

you're telling MPEX "the default mode of execution for the `%COBOL` and `%EDIT` commands is '\$' (i.e. offline) mode". Then, whenever you say

```
%COBOL...
```

this command will be executed just as if it was prefixed with a `"$"`. If you want to do it online (which would normally be the default), you'd have to say

```
%!COBOL...
```

telling MPEX to do the command in `"!"` (online) mode.

If you want to do `%SET DEFAULT`s, you probably want to include them into your `MPEXMGR.PUB.VESOFT`, `MPEXMGR.PUB.youracct` or `MPEXMGR.yourgroup.youracct` files — for more information, see the section ["Specifying commands to be executed every time MPEX is entered"](#).

Implied run

Syntax:

```
%programfile [.group [.account]]
    [[;INFO=] infostring]
    [[;PARM=] parmvalue]
    [;ENTRY=entrypoint]
    [;mperunparms] [...]
```

Using MPEX: Implied run

Examples:

```
%DBUTIL          << perhaps to run DBUTIL.PUB.SYS... >>
%AP010.PUB       << run a specific program >>
%MYPROG          << look in your own group first >>
%MYPROG TESTING  << pass an INFO= string >>
%MYPROG "X Y Z",10 << pass both an INFO= and a PARM= >>
                  << note ";PARM=" may be replaced by ",," >>
%MYPROG LIB=P;INFO="FOO";PRI=DS
```

If you type a command that is not a valid MPE/MPEX command or UDC, MPEX will then try to find a program or command file with that name and execute it as if you had typed `:RUN` or `:XEQ` at the beginning of the line. Say that you type:

```
%DBUTIL
```

(and presuming you do not have a UDC called "DBUTIL") MPEX will find the HP supplied program `DBUTIL.PUB.SYS` and evaluate the command as if you'd entered:

```
%DBUTIL.PUB.SYS
```

Which program file is run?

What do we mean by "try to find a program file with that name"? Well, if you specify a qualified filename, e.g.

```
%DBUTIL.PUB.SYS      or
%AP010.PUB
```

then MPEX will simply run the program with that name (just as if you'd put `"RUN "` at the beginning of your command).

Now, say that you specify a filename without a group or account, e.g.

```
%DBUTIL      or
%MYPROG
```

MPEX will then try to find a program with that name

- first in your logon group,
- then in the `PUB` group of your logon account,
- and finally in `PUB.SYS`.

If `DBUTIL` is found in your logon group, it'll be run; if `DBUTIL` is not found in your logon group or in your `PUB` group, but is found in `PUB.SYS`, then `DBUTIL.PUB.SYS` will be run.

Changing the search path

Of course, your natural question at this point must be "what if I want to always search some other groups as well?". What if you want to look in `PUB.SYS`, then in `UTIL.SYS`, and then in `PUB.PROD`?

No problem! Just say

```
%SETVAR HPPATH "!!HPGROUP,PUB,PUB.SYS,UTIL.SYS,PUB.PROD"
```

The HPPATH variable contains a list of groups to be searched, separated by commas.

The first parameter is "!!HPGROUP", which refers to the logon group (see the [MPEX Variables](#) section — it'll also explain why we put two "!"s instead of just one). The second parameter is PUB — the PUB group of the logon account; then come PUB.SYS, UTIL.SYS and PUB.PROD. The default setting of HPPATH is

```
%SETVAR HPPATH "!!HPGROUP,PUB,PUB.SYS"
```

Needless to say, you could delete some of these items as well as add new ones — saying

```
%SETVAR HPPATH "UTIL.SYS,PUB.PROD"
```

tells MPEX to search only in the UTIL.SYS and PUB.PROD groups.

Specifying ;INFO= and ;PARAM=

Simply saying

```
%MAIN.PUB.VESOFT
```

is the same as saying

```
%RUN MAIN.PUB.VESOFT
```

Similarly, saying

```
%MAIN.PUB.VESOFT HELP,1
```

or

```
%MAIN.PUB.VESOFT INFO="HELP";PARAM=1
```

is the same as saying

```
%RUN MAIN.PUB.VESOFT;INFO="HELP";PARAM=1
```

If you want to specify an entrypoint, simply use the ;ENTRY= keyword. You may also specify any other parameters — ;LIB=, ;MAXDATA=, ;STDIN=..., etc. — that you can specify on the %RUN command. (Note that this is different from the MPE/iX CI, which only lets you specify ;INFO= and ;PARAM=.)

Command files

Syntax:

```
%cmdfile [.group [.acct]]  
          [parmvalue] [...]  
          [parmname=parmvalue] [;...]
```

Examples:

```
%MYFILE 10, 20, 30
%MYFILE FRAMASTATS=40;THINGAMABOBS=10
```

In the "[Implied run](#)" discussion above, we said that if you enter a command that's not a valid MPE or MPEX command (or UDC), MPEX will look for a file with that name in

- your logon group,
- the PUB group of your logon account,
- and PUB.SYS.

If MPEX finds a program file, it'll :RUN it (just as if you'd entered a %RUN command). What if it finds a non-program file?

Well, if MPEX finds a normal ASCII file, it will assume that it is a COMMAND FILE, and will execute all the MPE commands stored in this command file. For instance, if you create a file called SCHEMA.PUB.SYS that contains the lines

```
PARM TEXTFILE, LISTFILE="$STDLIST"
FILE DBSTEXT=!TEXTFILE
FILE DBSLIST=!LISTFILE
RUN DBSCHEMA.PUB.SYS;PARM=3
```

and you then type

```
%SCHEMA APBASESC,*LP
```

MPEX will execute the commands

```
%FILE DBSTEXT=APBASESC
%FILE DBSLIST=*LP
%DBSCHEMA.PUB.SYS;PARM=3
```

In other words, a command file is very much like a UDC; it may have parameters, OPTIONS, defaults, etc.

The only differences are:

- The command file need not be :SETCATALOGed. Any command file that is in your search path (see the discussion of the [HPPATH](#) variable below) is available to you.
- The %SCHEMA command is implemented by the very existence of the command file SCHEMA.PUB.SYS.
- The header line (if any) should start with "PARM", e.g.

```
PARM TEXTFILE, LISTFILE="$STDLIST"
```

- rather than with the UDC name — the name of the command is already defined by the name of the command file. (The "PARM" line is not required. If you don't intend to pass any parameters to the command file, don't include a "PARM" line.) You can have multiple "PARM" lines, rather than trying to squeeze all of your parameters into one line.
- Naturally, each command file can define only one command (as opposed to UDC files, each of which may define many commands).

The advantages of UDCs are

- They support **OPTION LOGON** (both command files and UDCs support all the other options).
- They may be used for security purposes to block out some MPE commands.
- They are somewhat faster at command execution time, since they don't require a file open; on the other hand, they slow logon time.

Special security note

MPEX allows a user to execute a command file even if he only has eXecute (not Read) access to it. This way, you can create a command file that may contain sensitive information (such as lockwords) that you don't want a user to see. Although this is technically incompatible with MPE/iX, we believe that it is substantially more useful.

Command files and "HPPATH"

As we mentioned above, command files (just like program files or implied run commands) are looked for

- first in your logon group,
- then in the **PUB** group of your logon account,
- and then in **PUB.SYS**.

If you want to change this search sequence, you can use the same **HPPATH** variable used for the **IMPLIED RUN** searching, described earlier in this chapter.

ANYPARM and REST\$ parameters

Consider this: how would you implement the **MPE :FCOPY** command as a UDC (command files being pretty similar to UDCs)?

You might say

```
FCOPY !CMD
RUN FCOPY.PUB.SYS;INFO="!CMD"
```

but then if you type

```
:FCOPY FROM=A;TO=B;NEW
```

MPE will reject it — the **FCOPY** UDC (or command file) was declared with only one parameter, but you entered three ("**FROM=A**", "**TO=B**", and "**NEW**"). Since ";" (as well as ",", " " and to some extent "=") are considered by MPE to be delimiters, the parameter to the **FCOPY** UDC must not include any of these delimiters or else it must be enclosed in quotes.

What you need is some way of saying "consider the rest of this command to be a single parameter". For instance, you might say

```
FCOPY REST$CMD
RUN FCOPY.PUB.SYS;INFO="!CMD"
```

Using MPEX: Command files

The "REST\$" prefix in the parameter name indicates that this is a "rest-of-the-line" parameter. Note that the REST\$ is only specified in the header record, not every time !CMD is mentioned.

When you type

```
%FCOPY FROM=A;TO=B;NEW
```

MPEX will consider the entire parameter string — "FROM=A;TO=B;NEW" — to be a single parameter (CMD).

The REST\$ parameter must obviously be the last parameter of the command, but need not be the only one. You might, for instance, want to define a UDC such as

```
PROCESS FROMFILE, TOFILE, REST$PARMS  
FILE MYFROM=!FROMFILE  
FILE MYTO=!TOFILE  
RUN MYPROG;INFO="!PARMS"
```

Now, when you say

```
PROCESS A, B, C, D, E
```

the "FROMFILE" parameter will be set to "A", the "TOFILE" parameter to "B", and the "PARMS" parameter to "C, D, E". The REST\$ parameter is set to the REMAINDER of the command string after all the other parameters have been assigned.

Of course, this is available only in UDCs and command files that are executed from MPEX. Naturally, if you try to execute from MPE a UDC that includes a REST\$ parameter, you won't get very far.

Unfortunately, MPE/iX won't even let :SETCATALOG a UDC that has "REST\$" in the first line. However, MPE/iX has a feature (fully supported in MPEX) ANYPARM, that is quite similar to REST\$.

Note: While REST\$ can be the last parameter on the last PARM line, ANYPARM must be the last PARM line, all by itself.

For example:

```
PARM A, B, C  
ANYPARM D  
...
```

means that parameter "D" is to be set to the rest of the command line (just like "REST\$") — thus, if the above command file is passed a parameter string of "ALPHA, BETA, GAMMA, DELTA, EPSILON, ZETA", parm A will have the value "ALPHA", B will be "BETA", C will be "GAMMA" and D will be "DELTA, EPSILON, ZETA" (with the commas!).

Of course, like any other parameter, both REST\$ and ANYPARM parameters allow defaults (which makes them "optional"). However, if you want them to default to a null string (""), you must do one of the following:

```
PARM REST$FOOBAR=![""]
```

or

```
ANYPARM FOOBAR=![""]
```

If you just say 'FOOBAR=""', the variable **FOOBAR** will actually contain the quote characters.

Another feature that you may find essential when using **ANYPARM** and/or **REST\$** is our **CMDPARM(...)** function. Please see [Appendix B](#) for details.

CS\$ parameters

One other option we have for UDCs and command files is the **CS\$** option:

```
PARM CS$FILESET, NEWDEV
ALTFILE !FILESET; FLIMIT=EOF*2; DEV=!NEWDEV
```

What does this command file (let's call it **MOVETHEM**) do? (We know that it's a command file because its header starts with "PARM".)

Well, it takes two parameters — an MPEX fileset and a new device name — and **%ALTFILES** all the files in the fileset to

- have a file limit that's double their current number of records, and
- reside on the device indicated by **NEWDEV**.

So what's all this **CS\$** stuff? Well, say that we type

```
%MOVETHEM @.DATA(CODE="PRIV" and SECTORS>1000), 3
```

We're asking MPEX to move to device 3 the files in the **DATA** group that have code "PRIV" and use more than 1000 sectors.

This seems like a reasonable enough command; unfortunately, both "=" and " (quote) are considered by MPE to be parameter delimiters. If **%MOVETHEM** were a normal MPE UDC (or a normal MPE/iX command file), MPEX would reject the command we just showed unless the entire fileset parameter was enclosed in quotes.

The **CS\$** before the **FILESET** parameter in the command file header stands for "Comma/Semicolon". It says that the only delimiters that should be recognized for the parameter are Comma and Semicolon, not space or the equal sign. It also indicates that any delimiters inside parentheses (e.g. in selection conditions) are to be ignored.

As you can see, this feature was designed especially for MPEX filesets, and is most useful when you're passing MPEX fileset parameters.

Determining whether you're in MPEX or not

Sometimes, you may want to write a command file or UDC that works successfully from MPEX and in MPE/iX, but, when executed from MPEX, uses some special MPEX feature.

MPEX treats the JCW "INSIDEMPEX" as a special sort of JCW. When you enter MPEX, MPEX does a **":SETJCW INSIDEMPEX=0"**, but whenever you try to reference **INSIDEMPEX** from an MPEX command, MPEX will always return the value 1 (even though its "true" value is 0). This

Using MPEX: Redirecting command input and output

way, a command file or a UDC that is executed from within MPEX will see `INSIDEMPEX` as being equal to 1, but when executed from MPE, will see `INSIDEMPEX` as being equal to 0. Thus, the command file or UDC can say `"IF INSIDEMPEX=1 THEN"` or `"IF INSIDEMPEX=0 THEN"` to tell if it's in MPEX or not.

The only possible problem arises if the command file or UDC is executed from within MPE before the first run of MPEX in the session, in which case the `INSIDEMPEX JCW` will not be set, and an `IF INSIDEMPEX=1 THEN` will get a CI error. You can solve this by adding a `:SETJCW INSIDEMPEX=0` to the beginning of the command files/UDCs that you wish to test `INSIDEMPEX` within, either at the beginning or right before you test it. Then, when you test it in MPEX, MPEX will tell you it is 1. If you test it outside of MPEX, MPE will tell you it is 0.

Redirecting command input and output

MPEX lets you very easily redirect the input and output of any MPEX command, including command file execution, `%RUN`, etc.

Simply specify in the command

```
<infile,           e.g. %RUN MYPROG;INFO="XYZ" <DATAFIL  
>outfile,          e.g. %SHOWOUT JOB=@;SP >OUTDATA  
>>outfile (to append), e.g. %ECHO Execute step 5 >>LOGFILE
```

This is thus both a convenient shorthand for `;STDIN=...` and `;STDLIST=...` on `:RUN` commands and implied `RUN`, and a way of redirecting commands (like `%SHOWOUT` and `%ECHO`) and command files output that could not have been easily redirected before.

- The `<`, `>` and `>>` can be specified anywhere in the command, so long as the filename is followed by a blank or some such delimiter.
- If you specify `"<infile"`, MPEX will first try to use the temporary file with that name, and if that doesn't exist, then the permanent file. If you specify `">outfile"` or `">>outfile"`, MPEX will try to write to or append to the temporary file with that name, and if that doesn't exist, will create a new temporary file. If you want to output to a permanent file, you must use a `:FILE` equation with `",OLD"` (for old files) or `";SAVE"` (for new files). This might be somewhat inconvenient, but we have to do this to be compatible with MPE/iX.
- This syntax may present some problems, e.g. if you have existing commands that contain `<`, `>` or `>>` followed by valid-seeming filenames (which shouldn't be very common; we've noticed it most often in `ECHO` commands). The MPE/iX implementation of this feature suffers from similar problems.
 - To make it less likely that problems will occur, the commands `CALC`, `COMMENT`, `ELSEIF`, `IF`, `SETJCW`, `SETVAR`, `TELL`, `TELOP`, `WARN`, `WARNF`, `WHILE` and such do not allow input or output redirection (which wouldn't make sense for them anyway).
 - Also, `<`, `>` and `>>` are not resolved inside quoted strings, `![...]` constructs or (this is specific to MPEX, not MPE/iX) inside parentheses.

Note that the apostrophe is considered a quote character, so if you use a contraction in an echo statement, it won't work the way you might expect — MPEX sees the apostrophe as an opening quote, so everything until the next apostrophe or the end of the line is treated as "inside" a quoted string.

For example:

```
%ECHO This won't work >>workfile
```

will display everything to the screen and will not put the string "This won't work" into the workfile. In order to avoid this problem, MPEX allows you to specify the output file anywhere in the line, even as the first item, so to actually put a string with just one quote character into a file, use the following:

```
:ECHO >>workfile This will work, even with a ' character.
```

- If you need to embed a <, > or >> into a command, you may prefix them with a "!", e.g. "ECHO VAR1!>VAR2".
- By default (unless overridden with a file equation), the output files are built with variable-length records, no carriage control, and a file limit of 10,000 (which practically means that more than 10,000 variable-length records will fit into the file).
- By default, any command execution errors for an output-redirection command (except errors in opening the redirection files) will be sent to the output file. If you want them to go to the terminal, set the variable HPERRSTOLIST to TRUE (see also [Errors and warnings](#) in Appendix A).

Checking MPEX command success

Note: If you are interested in error-handling, you should also look at the MPEX %TRAPERROR command. It provides an advanced, easy to use, mechanism for handling errors in jobs, command files, etc.

How can you tell — in a job stream, in a command file or in a UDC — whether an MPEX command succeeded?

Well, if it is a normal MPE command (like :BUILD or :FILE) or one of our advanced fileset handling commands (like %RENAME or %PURGE) in which only a single file was specified, we'll set the CIERROR JCW as MPE normally would; we'll even set two JCWs:

- FSERROR, which indicates the file system error (if any) encountered while executing the command.
- CIERRORISWARN, which is 1 if the CI error was a warning and 0 if it was a true error.

Similarly, if we encounter a truly global error in one of MPEX's own commands — e.g. you specified a syntactically invalid fileset — we set the CIERROR JCW to one of our own error numbers (largely 30,000 and above). At the very least it will abort the execution of the command file, UDC or job; and, if you do an :IF CIERROR<>0 THEN or something like that, you'll see that an error occurred.

Now, say that you're doing a fileset %ALTFILE. 67 files were successfully altered but 4 couldn't, because of file system errors (security, file in use, etc.). What should we do?

Well, on "file-level errors" — errors that occur on a specific file — we do not abort the command file, UDC or MPEX job; in fact, we usually won't even set the above JCWs.

Using MPEX: Setting default parms for MPEX commands

However, we do provide you with two JCWs that tell you in great detail how successful the command was:

- MPEXNUMSUCCEEDED is set to the number of files successfully processed.
- MPEXNUMFAILED is set to the number of files on which errors were encountered.

With these two JCWs, you can do all the checking you want. You might say (in your job):

```
%IF MPEXNUMFAILED>0 THEN
%  TELLOP AT LEAST ONE ERROR OCCURRED!
%ENDIF
```

(to check if there was at least one failure) or

```
%LISTF @.@(CREATOR<>"MANAGER")
%IF MPEXNUMSUCCEEDED=0 THEN
%  TELLOP NO FILES CREATED BY NON-MANAGER USERS
%ENDIF
```

or even (if you want to get really fancy)

```
%COBOL AP@.SOURCE,AP@.PUB
%IF MPEXNUMFAILED>(MPEXNUMFAILED+MPEXNUMSUCCEEDED)/10 THEN
%  TELLOP COMPILE MORE THAN 10% BAD, CALL THE PROGRAMMER
%  TELLOP AND TELL HIM TO COME IN THIS WEEKEND.
%ENDIF
```

Setting default parms for MPEX commands

You can set any default parameters you want to for the %ABORTJOB, %ALTFILE, %ALTSCHED, %ALARM, %ALTJOB, %BREAKJOB, %BYE, %CALENDAR, %CHLOGON, %COPY, %DEVCONTROL, %DOSAVED, %IDENTIFY, %LISTJOB, %NEWLINK, %PAUSEJOB, %PREP, %PREPRUN, %PRINT, %PRINTO, %PURGE, %PURGELINK, %RENAME, %RESUMEJOB, %RUN (and implied RUN), %SAVEJOB, %SHOWALARM, %SHOWJOB, %SHOWME, %SHOWPROC, %SHOWSAVED, %SPOONFEED, and %SPOONINIT commands — e.g.:

```
%SETVAR VESOFTDEFAULTALTFILE ";KEEPAMDATES;XLTRIM"
%SETVAR VESOFTDEFAULTCOPY ";KEEPAMDATES;KEEPATTR"
%SETVAR VESOFTDEFAULTPRINT ";KEEPAMDATES;PAGEHEAD"
%SETVAR VESOFTDEFAULTCHLOGON ";KEEPCAPS"
```

To automatically kill any son processes that might suspend instead of terminating, set the following default:

```
%SETVAR VESOFTDEFAULTRUN ";KILL"
```

Warning: Setting this variable to this value globally (in MPEXMGR.PUB.VESOFT, for instance) will have adverse side effects on the operation of our software.

```
%SETVAR VESOFTDEFAULTPREP ";CAP=PH,DS;MAXDATA=30000;FMAP"  
%SETVAR VESOFTDEFAULTPREPRUN VESOFTDEFAULTRUN+VESOFTDEFAULTPREP
```

Sets %PREPRUN defaults to be same as %PREP defaults concatenated with the %RUN defaults.

```
%SETVAR VESOFTDEFAULTIMPRUN ";LIB=P;MAXDATA=30000"
```

For any programs run via implied :RUN, i.e. by just specifying the name of the program.

```
%SETVAR VESOFTDEFAULTBREAKJOB ";NOVERIFY"  
%SETVAR VESOFTDEFAULTRESUMEJOB ";NOVERIFY"
```

So that %BREAKJOB and %RESUMEJOB don't ask for verification.

The values of the VESOFTDEFAULTxxx variables must start with ";", as shown above.

If you want to use non-default settings, you certainly can — for instance, if VESOFTDEFAULTPREP is set to ";CAP=PH,DS;MAXDATA=30000;FMAP" and you do a :PREP...;CAP=PM, it will do the prep with PM capability and without PH and DS (since your ;CAP= overrides the default ;CAP=), but will still use the ;MAXDATA=30000 and ;FMAP.

If the default value that you want to override doesn't have an option that can turn it off (e.g. ;KEEPAMDATES), and you don't want to use this default in a particular command, you must delete the appropriate VESOFTDEFAULTxxx variable and then re-set it later.

Restricting MPEX commands to only work on certain files

Sometimes you know that certain MPEX fileset-handling commands don't make any sense (or can even do damage) when performed on certain types of files. For instance, the %EDIT command can't work on binary files because EDITOR can only handle ASCII files; furthermore, you might not want it to operate on KSAM files, since a /KEEP of a KSAM file will keep it as a "flat" file, which will preserve the data but destroy the KSAM file structure.

To implement such restrictions, just say

```
%SETVAR MPEXCRITRESTRICTEDIT "ISASCII AND NOT ISKSAM"
```

and the %EDIT command will behave as if you included the selection condition (ISASCII AND NOT ISKSAM) — i.e. select only ASCII files that aren't KSAM files — on the fileset. (Of course, this will also work if you explicitly specified a selection condition on the fileset; both conditions would then be used.)

If you want this sort of restriction to work for all users all the time, you can just put the %SETVAR into the MPEXMGR.PUB.VESOFT file; needless to say, a similar %SETVAR would work to impose selection conditions for other commands — just use %SETVAR MPEXCRITRESTRICT *cmdname*.

If you want to temporarily turn off this restriction, you can just %DELETEVAR the appropriate variable. If the %SETVAR is in MPEXMGR.PUB.VESOFT, the variable will be reinitialized the next time you enter MPEX.

MPEX VARIABLES

```
%SETVAR FNAME "MYFILE.PUB.TEST"

%SHOWVAR F@
FLUGELHORN = 172345
FNAME = MYFILE.PUB.TEST
FUNNY = TRUE

%CALC STR(FNAME,3,6)
FILE.P

%ECHO FILE = !FNAME.
FILE = MYFILE.PUB.TEST.

%DELETEVAR FNAME
%SHOWVAR F@
FLUGELHORN = 172345
FUNNY = TRUE
```

An MPEX variable — just like a COBOL or PASCAL variable — stores data.

MPEX variables:

- Are similar to JCWs, but may contain strings, boolean (true/false) values or 32-bit integers.
- Are set with the %SETVAR command.
- Can have their values input interactively using the %INPUT command.
- May be looked at with the %SHOWVAR command.
- Can be used in any MPEX expression, including:
 - %CALC/%SETVAR expressions,
 - %IF/%WHILE expressions,
 - and MPEX selection conditions.
- Can be substituted into any MPEX command file much like UDC parameters can be substituted into commands within UDCs.
- Stay around for the duration of your job or session (not just while you're running MPEX).

Using MPEX variables

MPEX variables, once they're set using the %SETVAR or %INPUT commands (which are documented in more detail elsewhere in this manual) can be used in one of several ways:

- MPEX variables can be used whenever MPEX expects an expression – in
 - %IF statements,
 - %CALC statements,
 - %WHILE statements,
 - %SETVAR statements,
 - MPEX fileset selection criteria, etc.

For instance, if CMDPARMS is a string variable, you might say

```
%IF STR(CMDPARMS,1,1)="X" THEN
```

thus using CMDPARMS as an element in the expression STR(CMDPARMS,1,1)="X".

- When MPEX does not expect an expression, you can substitute the value of a variable by prefixing it with an "!":

```
!!:BUILD !FNAME;REC=4095,,,ASCII
```

Here, if FNAME is a variable, its value is textually substituted in place of the string "!FNAME". If FNAME's value is XYZZY, the above !!:BUILD command is exactly identical to the command

```
!!:BUILD XYZZY;REC=4095,,,ASCII
```

Of course, this is very much like the way UDC parameter substitution has always worked in MPE.

- Instead of saying !VARNAME, you might also say !"VARNAME", e.g.

```
!!:BUILD Y!"MODULE"DAT;CODE=123;REC=-64,,,ASCII
```

As you see, the !"MODULE" is immediately followed by the alphanumeric string DAT — if we had said

```
!!:BUILD Y!MODULEDAT;CODE=123;REC=-64,,,ASCII
```

MPEX would have tried to find the value of the variable MODULEDAT rather than of MODULE. !"VARNAME" is thus identical to !VARNAME except that it can be used even if it's followed by an alphabetic or numeric character.

- An even more powerful construct is ![expression], e.g.

```
!!:BUILD !FNAME;DISC=![4*MAXUSERS+MAXPROGS]
```

The expression between the ![and the] — 4*MAXUSERS+MAXPROGS — is evaluated and is substituted for the entire ![4*MAXUSERS+MAXPROGS] construct. If FNAME were MYFILE, MAXUSERS were 100, and MAXPROGS were 25, the above command would be equivalent to

```
!!:BUILD MYFILE;DISC=425
```

The !xxx, !"xxx", and ![xxx] constructs are some of the most powerful features of MPEX variables, since they let you very easily parameterize MPEX commands — in UDCs, in command files or in jobs — to do exactly what you want them to do.

They make MPEX programming very easy and very powerful.

Setting MPEX variables

Of course, before an MPEX variable is used, it must first be set. Several ways are available for doing this:

- The `%SETVAR` command, which stores the value of an expression (or of a constant) in an MPEX variable, e.g.

```
%SETVAR MYFILE "XYZ.PUB.DEV"
```

or

```
%SETVAR MYFILE "S"+MODULENAME+".DEV.VESOFT"
```

(the second `%SETVAR`, as you see, sets `MYFILE` to the value of an expression that involves `MODULENAME`, another variable).

- The `%INPUT` command, which prompts the user for input, e.g.

```
%INPUT MODULENAME; PROMPT="Enter module name: "
```

- A number of MPEX variables are predefined; these include:
 - `HPUSER`, a string containing the logon user ID;
 - `HPLDEVIN`, an integer containing the device number of the logon terminal;
 - `HPINTERACTIVE` — `TRUE` in a session, `FALSE` in a job;
 - `HPUSERCAPF`, a string containing a list of all the capabilities a user possesses;
 - and many more.
- Some other variables are specially defined for MPEX purposes, and may be set and/or looked at by MPEX — see the [Special variables used by MPEX](#) section in this chapter.

Local variables

Often, within a command file, you will create variables that are only intended for use within that particular command file. You don't want these variables to conflict with existing, global variables you may have set (or with predefined HP or VESOFT variables), and you want to delete these variables after the command file is done executing. One way to do this is to precede the variable name with the name of the command file; i.e. in a command file named "FIDGET", you might use variables names `FIDGET_NAME`, `FIDGET_ADDRESS`, `FIDGET_PARM`, etc. Then, at the end of the command file, you could delete all of these variables with the command:

```
DELETEVAR FIDGET_@
```

But what you really want are local variables; and MPEX has them!

Local variables are quite similar to "regular", global variables; you create them with the `%SETLVAR` command (instead of `%SETVAR` — notice the "L"), delete them with `%DELETELVAR` (instead of `%DELETEVAR` — again, notice the "L"), and view them with `%SHOWLVAR` (instead of `%SHOWVAR` — there's that "L" again). Like global variables, local variables can be of type string, integer or boolean.

Local variables have three important properties:

- Unlike global variables, they are automatically deleted when the command file within which they were created terminates! (If you create a local variable by typing "%SETLVAR" directly within MPEX instead of in a command file, the variable will last until you exit MPEX).
- Local variables created in one command file are not accessible within any other command file! (Local variables created "directly" within MPEX do not exist within any command file at all.)
- When both a local and a global variables exist with the same name, the local variable will be used, e.g. the result of

```
%SETVAR MYVAR "global"
%SETLVAR MYVAR "local"
%ECHO the !MYVAR variable will be used.
The local variable will be used.
```

Note that this can be quite useful for testing. Say you have a command file that behaves differently in batch than it does online, by testing the variable HPJOBTYPE, and you want to observe the batch behavior while running it online.

If you try to SETVAR HPJOBTYPE "J", you'll get to see this error message:

```
ASSIGNMENT TO VARIABLE NOT ALLOWED: READONLY. (CIERR 8115)
```

But, if you add the line SETLVAR HPJOBTYPE "J" to the beginning of the command file, you can perform your test! And you don't have to worry about forgetting to delete the variable later so that it won't interfere with other command files — local variables (as we said above) are deleted automatically!

Other variable features

Other variable-handling commands — remember, all of them are documented in the MPEX COMMANDS section — include:

- %DELETEVAR, which deletes a variable or a set of variables.
- %SHOWVAR, which shows the values of a set of variables.
- %SET VARTRACE (and %SET NOVARTRACE), which shows all changes to the values of variables (useful in debugging).

Some other things you might want to know about variables are:

- If you're trying to look at the value of a variable (in an expression — e.g. in a %CALC command — or when you're substituting with ! or ![...]), and a JCW with that name exists, the value of that JCW will be used.

MPEX VARIABLES: Other variable features

For instance, if you say

```
%SETVAR TEMPERR CIERROR
```

or

```
%ECHO CI error #!CIERROR
```

then MPEX will retrieve and substitute the **CIERROR** JCW just as if it were an MPEX variable.

On MPE/iX, if you do an **%INPUT** or a **%SETVAR** into a variable whose name is that of an existing JCW, the JCW will be "changed" into a variable. On MPE/V the JCW will be set unless you are attempting to set it to a string, boolean or a value >65535, in which case you will get **CIERR 8135**. This is because MPEX variables are more flexible than JCWs, and may contain strings, 32-bit integers, and boolean values, where MPE JCWs can only contain 16-bit integers.

- Say that an MPEX string variable (one whose value is a string rather than an integer or a boolean) contains in its value an **!**, e.g.

```
%SHOWVAR FNAME  
FNAME = MYFILE.!HPGROUP.SYS
```

Then, whenever you substitute the variable with an **!** or a **!"..."** (but not a **![...]**), the variable will be recursively substituted. Say that your logon group is **JACK** (which means that the predefined variable **HPGROUP** would be set to **JACK**); when you say

```
%ECHO Filename = !FNAME
```

you won't just see

```
Filename = MYFILE.!HPGROUP.SYS
```

(which would be the case if MPEX merely substituted the value of **FNAME** for the string **!FNAME**), but will instead see

```
Filename = MYFILE.JACK.SYS
```

Instead of just having variables substituted once, variables were substituted repeatedly until no more substitutions were possible. Another example might be:

```
%SHOWVAR  
A = ABACUS  
B = B[!A]  
C = C/!B\  
D = D(!C)  
%ECHO D is !D  
D is D(C/B[ABACUS]\)
```

As you see, **!D** was recursively substituted until all the variable references — **!C**, **!B** and **!A** — were satisfied.

As you can also see, %SHOWVAR command does not do recursive substitution — neither do the ![...] construct or any expression evaluation commands (e.g. %CALC). Only ! and !..." do recursive substitution.

(If this confuses you, it's OK — it confuses me, too. I'm sure it could be quite useful in some cases, but it's pretty complicated. I really only put it in for compatibility with MPE/iX.)

A few comments about the "!" character

The fact that ! is a special prefix character (which causes a substitution of the expression that follows) creates some problems. If you say

```
%ECHO !HPUSER
```

then MPEX will output your user name (the value of the variable HPUSER). But, what if you really want to output the string "!HPUSER" — an exclamation mark followed by "HPUSER"?

To do this, we use the old MPE UDC convention of doubling the exclamation mark:

```
%ECHO !HPUSER
MANAGER
%ECHO !!HPUSER
!HPUSER
```

When MPEX reads a command and substitutes variables, it replaces all "!!"s by "!"s and does not substitute for whatever variable follows it.

This may seem simple enough, but it can cause a few interesting quirks. For instance, say that you enter

```
%TELLOP PLEASE MOUNT THE TAPE!!!!!
```

The message that the operator will see is

```
FROM/.../PLEASE MOUNT THE TAPE!!!
```

It includes only THREE exclamation marks. When you input "!!!!!", MPEX replaced each pair of exclamation marks by a single exclamation mark, thus changing exclamation marks 1 and 2 into a single "!", changing exclamation marks 3 and 4 into a single "!", and leaving exclamation mark 5 as a single "!" (since it is neither doubled nor has a variable name following it).

Got all that? Well, if you think you did, explain what is wrong with the command

```
%SETVAR MPEXPROMPT "!HPTIMEF: "
```

On the surface, it seems that you're setting the MPEXPROMPT variable so that MPEX will always prompt you with the current time — that way, at 1:35 PM, MPEX will prompt you with

```
1:35 PM:
```

MPEX VARIABLES: Special variables used by MPEX

and at 2:02 PM, MPEX will prompt you with

```
2:02 PM:
```

It seems that this should happen, but it won't. The first thing that MPEX does when executing a command is resolve all the "!"s. Thus, right after you enter (say, at 12:29 PM) the command

```
%SETVAR MPEXPROMPT "!HPTIMEF: "
```

MPEX will resolve the "!"s and turn the command into

```
%SETVAR MPEXPROMPT "12:29 PM: "
```

The variable "MPEXPROMPT" will be set to the constant string "12:29 PM: ". Thus, for the rest of day you'll be prompted with "12:29 PM: ".

What you should say instead is

```
%SETVAR MPEXPROMPT "!!HPTIMEF: "
```

Now, when MPEX resolves the "!"s, it'll translate the command into

```
%SETVAR MPEXPROMPT "!HPTIMEF: "
```

The variable MPEXPROMPT will be set to the string "!HPTIMEF: ". Then, when MPEX actually prints the variable, it'll resolve the "!"s again, prompting you with

```
1:35 PM:
```

Because the MPEXPROMPT variable was set to the literal string "!HPTIMEF: ", the HPTIMEF resolution isn't done until the prompt is actually printed (which is exactly the right time for it to be done).

Special variables used by MPEX

There are a number of very useful predefined variables (such as HPUSER, HPDATEF, HPINTERACTIVE, etc.) that are described in great detail in Appendix. They are largely read-only and return information about your current session or about the system environment.

There are a few other predefined variables that are specific to MPEX. Some of them are set by MPEX, but others can be set by you to control MPEX's behavior. Here they are:

Variable	Type	Usage
MPEXCMDTRACE	[INT]	(may be either a JCW or a variable). If this is set to 1, all commands that MPEX executes (including those that come from UDCs, from command files, from :WHILE loops, etc.) will be output to the terminal. This can be quite useful if you have some mysterious 'bug' in one of your complicated command files and you want to know exactly what's going on.

Variable	Type	Usage
		<p>MPEXCMDTRACE is not checked if a %SET NOCMDTRACECHECK is in effect (see the "%SET [NO]CMDTRACECHECK" command).</p> <p>Note: This JCW/variable can also be used to debug STREAMX jobs and SECURITY/3000 MENUs.</p>
MPEXPROMPT	[STR]	Indicates how MPEX is to prompt you for a command (default "%"). For more information, see " Changing the MPEX prompt " in the Running MPEX section.
HPSYSNAME	[STR]	Supplies a "system name" to include in the heading of all MPEX %LISTF output. Also used by SECURITY/3000's LISTLOG and LISTUSERS reports, and on VEAUDIT reports.
MPEXQEDITFILE	[STR]	<p>If you use MPEX's %QEDIT command (which executes an operation on a fileset using ROBELLE's popular QEDIT text editor), MPEX will try to run QEDIT.PUB.ROBELLE.</p> <p>If you've moved QEDIT out of PUB.ROBELLE, set the MPEXQEDITFILE variable to the location of QEDIT, e.g.</p> <pre>%SETVAR MPEXQEDITFILE "QEDIT.PUB.UTIL"</pre>
MPEXDBADGFILE	[STR]	<p>If you use MPEX's %DBADGALT command (which alters IMAGE database capacity on a fileset using the popular ADAGER program), MPEX will try to run ADAGER.PUB.REGO.</p> <p>If you've moved ADAGER to another group.account, set the MPEXDBADGFILE variable to the location of ADAGER, e.g.</p> <pre>%SETVAR MPEXDBADGFILE "ADAGER.PUB.TOOLS"</pre>
MPEXDBGENFILE	[STR]	<p>If you use MPEX's %DBGENALT command (which alters IMAGE database capacity on a fileset using BRADMARK's popular DBGENERAL program), MPEX will try to run DBGENRL.PUB.BRADMARK.</p> <p>If you've moved DBGENERAL to another group.account, set the MPEXDBGENFILE variable to the location of DBGENERAL, e.g.</p> <pre>%SETVAR MPEXDBGENFILE "DBGENRL.PUB.TOOLS"</pre>
MPEXTELLBACK	[STR]	<p>Tells MPEX who to inform when MPEX-created jobs (created with \$, %SUBMIT or %SCHEDULE) complete (or encounter errors).</p> <p>Setting this variable to "!!HPJOBNAME,!!HPUSER,!!HPACCOUNT" makes MPEX jobs send the message to all sessions logged on with your job/session name, your user ID and your account name.</p>

MPEX VARIABLES: Special variables used by MPEX

Variable	Type	Usage
		<p>Setting it to "#S! !HPJOBNUM" (the default) makes MPEX jobs send the message to your session specifically (if you log off and log back on, no message will arrive).</p> <p>If you want to, you can set this variable to anything you please – you might set it to "MANAGER.SYS" to make all jobs send messages to MANAGER.SYS.</p>
MPEXTELLBACKCMD	[STR]	<p>Normally, MPEX does a :TELL to notify you when your MPEX-created jobs (created with \$, %SUBMIT or %SCHEDULE) finish or encounter errors. You may use this variable if you would prefer a :TELLOP, %WARNF, or even have MPEX execute your own command file by setting it to something like "XEQ MYCMD".</p>
MPEXPREFIXDEF <i>nn</i>	[STR]	<p>"<i>nn</i>" is the ASCII code for any character (except for the reserved characters "%", "!", "?", "\$", ":" and ", "). Setting one of the variables lets you redefine special characters so that, when they are used as the first character of an MPEX command, they are replaced with a string.</p> <p>This feature is used primarily for "%SPOONFEEDing" (see %SPOONFEED in the commands reference section), but another useful example is</p> <pre>%SETVAR MPEXPREFIXDEF! [ORD('=')] "CALC "</pre> <p>which allows you to say things like</p> <pre>%=52*40 %=TODAY+30 %=VEFINFO("FOOBAR").CREDATE</pre> <p>etc.</p>
MPEXALARMPREFIX	[STR]	<p>Set this variable to a string that you want prefixed to all %ALARM messages you set.</p>
MPEXWARNFPREFIX	[STR]	<p>Set this variable to a string that you want prefixed to all %WARNF messages you send.</p>
VESOFTDEFAULT <i>xxx</i>	[STR]	<p>A group of variables that set defaults for certain MPEX commands. For more information, see "Setting default PARMS for MPEX commands" in the Using MPEX section.</p>
VECURREFILENAME	[STR]	<p>(read-only).</p> <p>This variable contains the name of the currently executing command file (in MPEX), job stream or ::USE file (in STREAMX) or menu file (in a SECURITY/3000 menu).</p>

FILENAMES AND FILESETS

MPE filenames

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

Examples:

```
MYFILE
JAN93.DATA
MAINPROG.SOURCE.DEVELOP
LOCKED/TIGHT.DATA.PROD
```

Nearly all MPEX commands deal with files and filenames in some manner, so a basic understanding of what defines a filename is needed in order to utilize our software in the best possible manner. Detailed information on files and filenames can be found in HP's manuals, so our discussion here will be very brief.

An MPE filename consists of file, group and account names. These names **MUST** begin with a letter and are followed by up to 7 letters or digits. No special characters are allowed, and the "." (dot) character separates file, group and account names. Group and account are optional; if not specified, they default to the user's logon group and account. Although you may use upper or lowercase letters, MPE translates MPE filenames to uppercase.

Lockwords are file access passwords. If you don't specify the lockword of a file when referring to it (such as when TEXTing in a source file in EDITOR), MPE will prompt you for the lockword. If you don't enter the correct lockword, access to the file will be denied.

MPE filesets

Syntax: `filespec [.groupspec [.accountspec]]`

Examples:

```
@.@.@
@.@.ACCOUNT
??@.GROUP.ACCOUNT
K#####.VER@.DOCUMENT
@ABC@.@DEF@
```

An MPE fileset is simply an MPE filename with optional wildcard characters. (Note that the LOCKWORD of the file is not specified for a fileset.)

These wildcard characters may be used:

Wildcard	Interpretation
@	Zero or more characters, either letters or digits
?	Exactly one character, either a letter or a digit
#	Exactly one digit

Wildcard	Interpretation
[...]	Exactly one character taken from the set of characters given between the square brackets. For example, "[123ASDF]" means any one of the characters "1", "2", "3", "A", "S", "D" or "F". A character range can be specified using the "-" (hyphen) character: "[A-C]" means either the letter "A", "B" or "C". Of course, these two can be combined, so "[A-HO-Z]" means any letter except "I", "J", "K", "L", "M" or "N".

POSIX filenames

Syntax:

```
.[/]posixfilename
/posixfilename
```

Examples:

```
.MYFILE
.myfile
/accounting/93_04_22_datafile
./subdir/ACTUAL_filename
/a_posix_file/with_a_very_long.but_still-legal.filename
/MPEACCT/MPEGROUP/MPEFILE
```

With the introduction of version 5.0 of the MPE/iX operating system, you are allowed to specify files using what is known as HFS syntax. "HFS" stands for "Hierarchical File System" and is often referred to as "POSIX", although "POSIX", as such, actually encompasses a lot more than the way you specify filenames. In this manual, you may see references to either HFS or POSIX, you can consider them the same unless noted otherwise. Of course, if you are not using a system that supports POSIX, then you cannot specify files using HFS syntax.

A POSIX filename, unlike an MPE filename:

- may contain upper AND lowercase letters (which are significant, /FILE is not the same as /file).
- may contain these special characters: "-" (hyphen), "_" (underscore), and "." (dot).
- may begin with any character except "-", which allows digits, ".", and the "_".
- may be up to 16 characters in length if the file is built in the root directory (beginning with just "/") or if it is built in any MPE group (usually anything beginning with ". /" if you have not changed your current working directory to a POSIX directory).
- may be up to 255 characters in length if the file is built anywhere else.
- can not have a lockword since the "/" character is used to delimit subdirectories.

Since MPE filenames are always in uppercase, MPE will automatically upshift any filename beginning with a letter from "a" to "z". In order to specify a POSIX filename to MPEX or any other MPE utility, the filename is prefaced with either a period, a forward slash or both.

POSIX files that begin with the forward slash character (/) are considered to be in the root directory, files that begin with a period (.) are in your current directory, and files that begin with two periods and a forward slash (./) are in your parent directory.

Forward slashes in the rest of a POSIX filename are used to delimit subdirectories, which are logically equivalent to MPE groups in an MPE account. (In fact, files in the MPE directory can be referred to using POSIX notation as shown in the last example.)

POSIX filesets

A POSIX fileset, like an MPE fileset, is an explicit filename that may contain wildcard characters. In MPE, POSIX filesets ending with a forward slash will include subdirectories recursively, so the POSIX fileset of "/" is equivalent to "@.@.@" for MPE files. POSIX filesets ending with an "@" character will not search any subdirectories for files, so the POSIX fileset of "/@" is the list of files, directories and accounts that are ONLY in the root directory.

The same wildcard rules apply for POSIX filesets as for MPE filesets with one notable exception. Since the "-" character is a legal character in a filename, and when specifying a range of characters such as "[I-N]" you have to use the "-" character as part of the syntax and not a legal character, it would seem impossible to specify a "-" character in a "[...]" range set.

As it turns out, if the "-" character is the first or last character in a range set, then it is treated as "one of the characters to allow" instead of indicating "a range of characters to allow". For example, to indicate that any letter from "I" to "N" OR the "-" character may be used, you can specify this range set as follows:

```
[-I-N]
```

or

```
[I-N-]
```

MPEX filesets

Syntax:

```
mpefileset [{+|-}] mpefileset [...] [(selectioncondition)]
posixfileset [{+|-}] posixfileset [...] [(selectioncondition)]
eitherfileset [{+|-}] eitherfileset [...] [(selectioncondition)]
```

Examples:

```
@.@.MYACCT
/posixdir/mydir/(SECTORS>1023)
@.@.SYS + @.@.TELESUP
/SYS/ + @.@.TELESUP - /@/PUB/@/
@.@.@(SAVABLESECTORS>100)
@.MYGROUP.MYACCT (ISHIDDEN)

@.@.@-@.PUB.SYS-@.@.VESOFT(ACCDATE<TODAY-120 and NOT ISPRIV)
/ - /SYS/ -/VESOFT/(ACCDATE<TODAY-120 and NOT ISPRIV)
```

One of the most important features of MPEX — probably the most important — is its ability to work on filesets. You can compile filesets, edit them, copy them, purge them, rename them, print them, etc.

However, the filesets MPEX can operate on aren't just ordinary MPE or POSIX filesets. They are a vastly enhanced superset of MPE and POSIX filesets, which includes features like:

- The ability to specify several filesets in one command, e.g.

```
%PURGE K#####.@.@ + LOG####.PUB.SYS
%PURGE K#####.@.@ + /SYS/PUB/LOG####
```

The "+" sign means "use all files in fileset 1 and all files in fileset 2". On systems that support POSIX, MPE filesets and POSIX filesets may be intermixed in the same command.

- The ability to exclude filesets, e.g.

```
%COPY @.DATA+@.SOURCE - MYDB@.DATA - COPY@.SOURCE, @.@.NEWDEV
%COPY ../DATA/+/../SOURCE/ - ../DATA/MYDB@ - ../SOURCE/COPY@,&
@.@.NEWDEV
```

The "-" sign means "use all files in the + filesets except for those files after the -".

POSIX note: Since "-" characters may form part of a POSIX filename, there must be a leading space before the "-" character if the preceding fileset is a POSIX fileset. You may find it convenient to always insert spaces around the "-" character in a fileset.

- The ability to select files by their file code, their last access date, the disk space they use, etc. — by virtually any file attribute you can think of. (This "*selectioncondition*" is a boolean expression; please see [Appendix B](#) for more details on expressions.)

For instance,

```
%PURGE @.@.@ - @.@.VESOFT (ACCDATE<TODAY-120 and NOT ISPRIV)
%PURGE / - /VESOFT/ (ACCDATE<TODAY-120 and NOT ISPRIV)
```

will purge all the files in the system (except those in the VESOFT account) that haven't been accessed in the last 120 days and have a filecode that's not "PRIV".

Other examples of this extraordinarily powerful feature include:

```
%LISTF @.@(ONDEVICE(1)), 4
```

which finds all the files that are at least partially on disk device 1, and:

```
%LISTF @.@.@(ISPRIV and (DBSETTYPE="A" or DBSETTYPE="M") and DBSETFULLNESS>.75),DB
```

which shows information on all the IMAGE Automatic or Manual master datasets (DBSETTYPE="A" or DBSETTYPE="M") that are more than 75% full (DBSETFULLNESS>.75).

```
%LISTF @.MYGROUP.MYACCT (ISHIDDEN), 2
```

The ISHIDDEN attribute will select only those files that are "hidden" from MPE's :LISTF output. Files that are "hidden" are files that begin with either a "/" or "." and contain characters that cannot be used in an MPE filename. (in other words, all of the files that can ONLY be specified using HFS notation.) This would include files that have more than 8 characters in the filename, contain lowercase letters or the "_", "-" or "." characters, or reside in a SUBDIRECTORY instead of a GROUP. Remember, MPEX's %LISTF..., 2 will display all MPE and HFS named files that qualify, even if only given an MPE fileset. Some

attributes that are only available on POSIX systems, such as BYTE STREAM files, can still have MPE-compliant names and therefore will not be "hidden" files.

+ filesets

"+ filesets" — filesets that are combined using the + operator are quite straightforward. A command such as

```
%PURGE K#####.@.@ + LOG####.PUB.SYS
```

simply operates on all the files in the first fileset (in this case, all EDITOR's K-files in the system) and on all the files in the second fileset (in this case, all the system log files). Other examples might include

```
%PRINT AP010.SOURCE + AP030.SOURCE + AP175.SOURCE + AP250.SOURCE; &  
OUT=*LP; PAGEHEAD
```

(which simply saves you having to type four separate %PRINT commands), or

```
%EDIT AP@.SOURCE + GL@.SOURCE, SET VARIABLE;LIST 1/2
```

which performs the given EDITOR command(s) on all the files in the SOURCE group that start with AP or GL.

The operations on the files are performed one fileset at a time, each fileset in the order specified. In other words, if you say

```
%LISTF @.SOURCE+@.DEV, 3
```

then the SOURCE files will be output first and the DEV files after them (even though the files within each fileset will be output in alphabetical order).

Similarly, if a file is included in more than one of the "+"ed filesets, it'll be processed each time it appears.

- filesets

"- filesets" allow you to EXCLUDE files from an MPEX fileset. For instance,

```
%COPY @.DATA+@.SOURCE - MYDB@.DATA - COPY@.SOURCE, =.NEWDEV
```

will copy into the NEWDEV account all the DATA and SOURCE group files (@.DATA+@.SOURCE) EXCEPT for those in the filesets MYDB@.DATA and COPY@.SOURCE.

Note that - filesets apply to the ENTIRE MPEX fileset, not just to the + fileset immediately preceding.

For instance, if you say

```
%LISTF A@+B@-@X-@Y
```

then ALL files that end with X or Y will be excluded (the "-@X-@Y" applies to "A@+B@", not just to B@).

Indirect files

MPEX lets you select files by their filenames (with wildcards, + filesets, and – filesets) and by various other attributes. What if, however, you want to operate on a list of files, which don't necessarily have much in common?

For instance, say that the source files for one of your programs are stored in the files `MAINLINE.SOURCE.AP`, `UTILSUBS.SOURCE.AP`, `XFILE.INCLUDE.AP`, and `DECLS.SOURCE.AP`. You might want to be able to compile them all, edit them all, copy them all, etc., without having to specify the fileset `MAINLINE.SOURCE.AP + UTILSUBS.SOURCE.AP + XFILE.INCLUDE.AP + DECLS.SOURCE.AP` on each `%COBOL`, `%EDIT`, `%COPY`, etc. command (imagine what the fileset would look like if these weren't just four files, but 40 or 400!).

Instead, you can build an editor file that contains 4 lines:

```
MAINLINE.SOURCE.AP
UTILSUBS.SOURCE.AP
XFILE.INCLUDE.AP
DECLS.SOURCE.AP
```

Say that you call this file `FILELIST.SOURCE.AP`; you could now say

```
%COBOL ^FILELIST.SOURCE.AP (MODDATE>TODAY-3), MYPROG
```

or

```
%PRINT ^FILELIST.SOURCE.AP;OUT=*LP;PAGEHEAD
```

or

```
%EDITCHG ^FILELIST.SOURCE.AP, "ITEM-NO", "ITEM-NUMBER"
```

Note the "^" before `"FILELIST.SOURCE.AP"`. This tells MPEX that this command (`%COBOL`, `%PRINT`, `%EDIT` or whatever) should not be executed on the file `FILELIST.SOURCE.AP` itself (which is what would happen if you'd omitted the "^"). Instead, it should be executed on the files listed in the file `FILELIST.SOURCE.AP`.

This "indirect file" can contain either filenames or filesets; it might, for instance, say

```
MAINLINE.SOURCE.AP
UTILSUBS.SOURCE.AP
X@.INCLUDE.AP
DECLS.SOURCE.AP
```

which means that it includes the three explicitly specified files (`MAINLINE`, `UTILSUBS`, and `DECLS`) but also all the files in the fileset `X@.INCLUDE.AP`. In general, you should think of an indirect file as just a single MPEX fileset with all the component files "+"ed together.

Each line of this file may be a filename or fileset, with @, # or ? wildcards and [...] range sets. On systems that support POSIX filenames, POSIX filesets may be included as well (in fact, both types of filesets may be in the file at the same time). However, it may not include "+"s, "-"s

(except as part of a POSIX filename) or selection criteria — features which are not normally present in ordinary MPE filesets.

You may also specify an indirect file as a "-" fileset, e.g.

```
%PURGE @.SOURCE-^FILELIST.SOURCE.AP
```

This is the same thing as specifying each line in the FILELIST file as a separate "-" fileset on the %PURGE command, i.e.

```
%PURGE @.SOURCE-MAINLINE.SOURCE.AP-UTILSUBS.SOURCE.AP- &
X@.INCLUDE.AP-DECLS.SOURCE.AP
```

Note that although you can have many hundreds of lines in an indirect fileset used for "+" purposes, you can have only up to about 80 lines in an indirect fileset being "-"d (the exact maximum depends on the length of each line).

If you like, you can include comments in indirect files by putting a "#" in column one (this feature was added for compatibility with the MPE/iX Link Editor). For example:

```
#This indirect file lists all of the NMOBJ files that need to be
#included in the production RL.
LIB1OBJ
LIB2OBJ
LIB3OBJ
LIB4OBJ
LIB5OBJ
```

Selection criteria

Probably the most useful aspect of MPEX filesets is the ability to select files based on some file attribute(s). For instance, you might say

```
%PURGE @.@.@ - @.@.VESOFT(ACCDATE<TODAY-120 AND NOT ISPRIV)
%PURGE /      - /VESOFT/ (ACCDATE<TODAY-120 AND NOT ISPRIV)
```

to purge all the files that match the selection criteria ACCDATE<TODAY-120 AND NOT ISPRIV (i.e. they haven't been accessed in 120 days AND their filecodes are not "PRIV").

Other examples include:

```
%LISTF @.@.@ - @.@.VESOFT&
          - @.PUB.SYS(PROG.PMCAP),5
%LISTF / - /VESOFT/&
          - @.PUB.SYS(PROG.PMCAP),5
```

which finds all the program files that have PM capability (PROG.PMCAP) except those that are in the VESOFT account or in the PUB.SYS group.

```
%ALTFILE @.@(CREATOR="JOHN"); CREATOR=MANAGER
```

which changes to **MANAGER** the creator ID of all the files in the logon account that were created by **JOHN** (**CREATOR="JOHN"**).

```
%LISTF CUST?#.DB(DBSETTYPE<>"D" AND DBSETFULLNESS>.65),DB
```

which shows all the **CUST** database datasets (**CUST?#.DB**) that are not details (**DBSETTYPE<>"D"**) and are more than 65% full (**DBSETFULLNESS>.65**).

```
%ALTFILE @.DATA(DEVICESECTORS(1)>=SECTORS/10); DEV=2
```

which moves to device 2 all the **DATA** files that have at least 1/10th of their disk space on device 1.

As you can see, the selection criteria can be arbitrary MPEX expressions, involving:

- File attribute variables and file attribute functions, which contain information about the current file.

For instance,

- **CREATOR** is a string variable containing the file's creator ID;
- **ACCDATE** is a date variable containing the file's last access date;
- **SECTORS** is an integer variable containing the number of disk sectors the file occupies;
- **PROG.PMCAP** is a boolean (TRUE/FALSE) variable that's TRUE if the current program file has PM capability;
- **ONDEVICE (*ldev*)** is a function that returns TRUE if at least one extent of the current file is located on the specified *ldev*;

Note that these variables and functions refer to attributes of the current file being processed (the file for which selection criteria are being checked). This should be distinguished from ordinary MPEX variables (set with **%SETVAR**) or operators/functions (like **+**, **-**, **UPS**, **STR**, etc.) that don't care about what the current file is.

- Standard MPEX variables, operators, and functions, such as **+**, **-**, **AND**, **OR**, **HPUSER**, **HPGROUP**, etc. These are all discussed in Appendix, and are the same in selection conditions, in the **%IF** statement, in the **%CALC** command, etc.

With all the operators and file attribute variables that are defined, you can select files on virtually any basis you can think of.

File ranges

What if you want to find all the files whose filenames are in a range; e.g. all the files that start with "K" through "P" in the **DATA** group? Well, you could say **%LISTF K@.DATA + L@.DATA + M@.DATA + N@.DATA + O@.DATA + P@.DATA** but (of course) there is an easier way.

In addition to the old MPE wildcards @, ?, and #, MPEX supports a file range wildcard, best described by example:

[ABC]	exactly one character, an A, B or C
[A-M]	exactly one character, an A, B, C, ..., L or M
[A-EP-SZ]	exactly one character, an A, B, C, D, E, P, Q, R, S or Z
[A-M]@	any number of characters that start with an A through M
@[05]@	any number of characters that contain a 0 or a 5

As you see, a "[...]" construct matches exactly one character, which must be one of the ones listed inside the brackets; saying "[x-y]" indicates a range of characters, and several ranges and/or single characters may be included in one "[...]" construct. So, for the example above, you would use the following command:

```
%LISTF [K-P]@.DATA
```

POSIX Since the "-" character is a legal character in a filename, but in the above syntax a "-" indicates
note: a range of characters, it would seem impossible to indicate that a "-" is simply "one of the allowed characters". However, if the "-" is the first or last character in the set, then it means that the "-" character is one of the allowable characters and not an indication of a range of allowable characters.

POSIX Since POSIX filesets are case sensitive, [a-z] is not the same as [A-Z]. To select a single
note: alphabetic character, without regard to case, use [a-zA-Z].

This syntax makes it easy for patterns you enter as parts of filesets, usersets, and MPE account-/user-/group-sets to exceed 8 characters; therefore, we've increased the maximum length of any part of such an object (i.e. the file, group or account part of a fileset), when used as part of a pattern, to 16 characters.

What if you want to find all the files whose filenames are in a range like AP050 through AP230? This is a commonly asked-for feature, and one that ordinary MPE wildcards can't handle (AP@ refers to all files that start with AP, and AP[0-2]@ isn't quite right, either, since it would include filenames like AP023 and AP256 — you just want those from AP050 to AP230).

MPEX's powerful selection conditions do the trick. Just say

```
%PRINT @(BETWEEN(FILE,"AP050","AP230"))
```

This selects all the files in your group (@) whose filenames (the file attribute variable FILE) are between "AP050" and "AP230". The BETWEEN function (documented in Appendix) will, in general, check to see if the value of its first parameter is between the values of its second and third parameters; file ranges are just a special case.

Of course, you can do this for any command, not just %PRINT... Also, in this command you could have said

```
%PRINT AP@(BETWEEN(FILE,"AP050","AP230"))
```

since all the files you want are also in the fileset AP@.

Target filesets

The filesets we've just been describing — the ones with +, -, and selection criteria — are called **SOURCE FILESETS**, filesets that can be specified as the first parameter of a fileset-handling command. A **TARGET** fileset is a fileset that is specified as the second or last parameter of a fileset-handling command. For instance, you might say

```
%COPY @.SOURCE (ACCDATE>=TODAY-2) ,@.CHANGED
```

in which case @.SOURCE (ACCDATE>=TODAY-2) is the source fileset, and @.CHANGED is the target.

What if you were to type this instead:

```
%COPY @.SOURCE , @.CHANGED-X@.CHANGED (ACCDATE>=TODAY-2)
```

Does this make any sense? Not really. The second parameter — the "target" of the %COPY command — doesn't specify which files are to be copied, but rather what the new names of the copied files will be. What does it mean when we put an "(ACCDATE>=TODAY-2)" selection condition on the target fileset? The object files need not even exist (they probably don't) — how can we select based on their last access date? And what about the "-X@.CHANGED" — what if one of the files in SOURCE starts with X? If we can't copy it into a file that starts with X in the CHANGED group, what could we do?

What this all means is that + filesets, - filesets, and selection conditions work only on source filesets (the first fileset on each fileset-handling command). Target filesets, with one minor exception, must be simple MPE or HFS style filesets.

The exception for target filesets is the "=" wildcard character. Say that you type

```
%COPY AP@.SOURCE+GL@.SOURCE , =.BACKUP
```

The "=" sign means "make the filename of the target file be the same as the filename of the source file". (If you'd put the "=" in the group, account or lockword position, it would make the group,account and lockword of the target be the same as the group,account and lockword of the source.)

How is an "=" different from an "@"? Well, "=" refers to the **ENTIRE** source file part (or group or account) — "@" refers only to that part which corresponds to an "@" in the source fileset. In other words, saying

```
%COPY S@.DEV.AP , B@.DEV.AP
```

will copy the file STES.T.DEV.AP into the file BTEST.DEV.AP — the "@" in the target fileset was replaced by "TEST". Saying

```
%COPY S@.DEV.AP , B=.DEV.AP
```

will copy the file STES.T.DEV.AP into the file BSTES.T.DEV.AP — the "=" in the target fileset was replaced by "STES.T", the entire file portion of the source file name.

"=" is most useful in cases like those shown in the first example:

```
%COPY AP@.SOURCE+GL@.SOURCE, =.BACKUP
```

Here, the target filename is defined to be the **SAME** as the source filename, regardless of which fileset (AP@.SOURCE or GL@.SOURCE) the source file came from. If the source fileset didn't have a "+", e.g.

```
%COPY AP@.SOURCE, =.BACKUP
```

then we could easily do the same thing without an "=":

```
%COPY AP@.SOURCE, AP@.BACKUP
```

Another important application of "=" is to preserve file lockwords on a %COPY, %FCOPY or %RENAME:

```
%COPY @.SOURCE, @/= .BACKUP
```

The "=" in the lockword part of the target fileset means that the target file should be built with the same lockword (if any) as the source file. Otherwise, if you don't specify a lockword, MPEX will assume (just like MPE) that the target file should be unlockworded.

POSIX Note: source and target filesets using HFS syntax

On systems that support POSIX, certain changes have to be made for special characters in a source or target fileset. For instance, we have already mentioned that the "/" character is used to delimit directories and subdirectories and therefore cannot be used to indicate a lockword in a POSIX fileset. Likewise, since the "-" character is a legal character in filenames, (but not as the first character of a filename), you must insert a space before a file or fileset that is to be subtracted if the previous file is specified using HFS syntax.

The "@" character still means "zero or more characters", but it will not go beyond a "/" character (which, in this case, is the same as the "." character used to separate the file, group and account parts of an MPE filename). Like the "@", the "=" character in a target fileset will replace the current element with the complete corresponding element from the source fileset. This is fine in most cases, but could be a bit tedious if you are renaming or copying a file that is several directory levels deep. In this case, the shorthand notation of "==" in a TARGET fileset means "use the same path as the source" (see [below](#) for an example).

While the standard wildcard characters will let you specify nearly any fileset that you are likely to ever use, there are still some filesets that cannot be specified easily. For instance, the fileset:

```
/A@/B@/C@
```

is fairly straightforward — it selects any file that begins with the letter "C", residing in any subdirectory beginning with a "B", located in any directory beginning with an "A".

However, the file:

```
/ARIZONA/BUSINESS/TAXES/COLLECTIONS
```

will not qualify since "COLLECTIONS" is in the fourth level while "/A@/B@/C@" only has three levels.

In order to select files in any sub-level, you can use "/" to mean "any number of directories and subdirectories". So, in order to qualify "COLLECTIONS" in the above example, use:
/A@/B@/C@

However, the "/" may not be followed by another "/" or in other words, the only thing that can follow a "/" is a wild-carded FILE specification, not a wild-carded **DIRECTORY** specification.

In both of these examples, the last element of the source fileset was "C@", which will only select files and directories that start with "C" in the directory specified. If the last character of a POSIX source fileset is a "/", then MPEX will include all the files in each subdirectory that matches the last directory level specified. For example, let's say you have a directory called "/source", with several subdirectories of "project-01", "project-02", and so on. In each "project-##" directory, the actual source code, "include" files, libraries, notes, and any other related files are stored in one or more subdirectories. In order to specify "everything related to project 11", you could use the fileset:

```
/source/project-11/
```

To specify all files for all projects, you would use:

```
/source/project-@/
```

Since an HFS pathname can contain several levels, it could be a bit tedious to have to type "/=/=/=/=/=/=/NEWTARGETNAME" when renaming or copying files within the same directory, so for POSIX target filesets, the shorthand notation of "==" means "the same path as the source". For example, to rename a file that is buried several levels deep (keeping the file in the same directory) the following syntax can be used:

```
%RENAME /level1/level2/level3/level4/level5/filename, ==/newfilename
```

If the **TARGET** of a copy or rename operation is a directory, then all the files from the source fileset are copied or renamed into the directory (This is similar to other operating systems such as UNIX or MS-DOS).

File attribute variables and functions

As we mentioned before, a selection criterion is an expression that can include some **FILE ATTRIBUTE VARIABLES AND FUNCTIONS** that contain information about the current file.

In general, whenever MPEX operates on a fileset, it makes these file attribute variables/functions available to you so that you can make decisions based on the attributes of the current file.

This includes:

- Selection criteria,
- %LISTF command flexible template files,
- the VEFINFO(...) .attrvar function, and
- %REPEAT/%FORFILES statements, in which you can refer to file attribute variables and functions by using the ! [xxx] variable syntax and prefixing each file attribute variable name with "RFILE." — e.g.

```
%REPEAT
%>FILE MYINFILE=!MPEXCURRENTFILE
%>RUN MYPROG;INFO="![RFILE.CREATOR]";PARAM=! [RFILE.INTCODE]
%>FORFILES @.DATA
```

Each file attribute variable — like all MPEX variables — has a data type (INTEGER, REAL, STRING or BOOLEAN). Unlike MPEX variables but like some MPEX operators (see [Appendix B](#)), some attribute variables might also be of type DATE or type TIME.

It's important to know the variable's type because the type controls what operators you can use on the variable — for instance, if you say %LISTF @.@.(ACCDATE<CODE) ,3 you'll get an error — ACCDATE is of type DATE, CODE is of type STRING, and you can't compare dates and strings. On the other hand,

```
%LISTF @.@.(ACCDATE<TODAY-120) ,3
```

is quite valid:

- ACCDATE is a file attribute variable of type DATE.
- TODAY is an MPEX function (see [Appendix B](#)) of type DATE.
- The – operator can take a date (TODAY) and an integer (120), and will return another date.
- Finally, you'll be comparing ACCDATE (a date) against TODAY-120 (also a date), which is perfectly legal.
- With all that said, here are the file attribute variables and file attribute functions that MPEX defines.
- Remember, you can use them directly in selection criteria and %LISTF template files — in %REPEAT...%FORFILES constructs they have to be prefixed with "RFILE.".
- Also remember that all of these attribute variables/functions apply to THE FILE BEING CURRENTLY PROCESSED. Thus, when we say that "ACCOUNT" refers to "Account name", we mean the account name of the current file.
- Each variable is marked [STR] if it's a string, [INT] if an integer, [BOOL] for boolean, [DATE] for date, [TIME] for time, and [REAL] for real.
- [STR] variables always have trailing blanks removed.
- For your convenience, the file attribute functions and variables are roughly categorized by the LISTF mode (if any) that shows them — all the things shown by %LISTF... ,2 first, then those shown by %LISTF... ,XL3, etc. Of course, each file attribute can be used in any MPEX command, not just %LISTF.

```

:LISTF MYFILE.PUB.SYS,2
ACCOUNT=  SYS          GROUP=  PUB

FILENAME  CODE  -----LOGICAL RECORD-----  ----SPACE----
          SIZE  TYP          EOF          LIMIT R/B  SECTORS #X MX
MYFILE   * EDTCT   80B  FA          14100          18901  32          5930  27  29

```

Attribute	Type	Returns	Example
FILE	[STR]	Filename	"MYFILE"
GROUP	[STR]	Group name	"PUB"
ACCOUNT	[BOOL]	Account name	"SYS"
OPENED	[BOOL]	TRUE if file currently FOPENed	TRUE
FMTOPENED	[STR]	"*" if FOPENed, " " if not	"*"
INTCODE	[INT]	Filecode as an integer	1052
CODE	[STR]	Filecode as a string	"EDTCT"
RECSIZE	[INT]	Record size in bytes	80
FMTRECSIZE	[STR]	Record size	80B
FMTTYPE	[STR]	File type	"FA"
EOF	[INT]	Number of records in file	14100
FLIMIT	[INT]	File limit	18901
BLOCKFACTOR	[INT]	Blocking factor	32
BLOCKSIZE	[INT]	Block size, in bytes	2580 (80*32)
SECTORS	[INT]	# of sectors used by file	5930
NUMEXTENTS	[INT]	Current number of extents	27
MAXEXTENTS	[INT]	Maximum number of extents	29
ISFIXED	[BOOL]	TRUE if fixed length records	TRUE
ISVARIABLE	[BOOL]	TRUE if variable records	FALSE
ISUNDEFINED	[BOOL]	TRUE if undefined records	FALSE
ISASCII	[BOOL]	TRUE if file is ASCII	TRUE
ISBINARY	[BOOL]	TRUE if file is BINARY	FALSE
ISCCTL	[BOOL]	TRUE if file has CCTL	FALSE
ISNOCCTL	[BOOL]	TRUE if file has NOCCTL	TRUE
ISSTD	[BOOL]	TRUE if not MSG/CIR/RIO/KSAM file	TRUE

Attribute	Type	Returns	Example
ISRIO	[BOOL]	TRUE if type RIO file	FALSE
ISKSAM	[BOOL]	TRUE if KSAM (or KSAMXL) key/data file	FALSE
ISKSAMXL	[BOOL]	TRUE if KSAMXL file	FALSE
ISKSAM64	[BOOL]	TRUE if KSAMXL64 file	FALSE
ISMSG	[BOOL]	TRUE if MSG file	FALSE
ISCIR	[BOOL]	TRUE if CIRcular file	FALSE
ISCMPROG	[BOOL]	TRUE for PROG files only	FALSE
ISNMPROG	[BOOL]	TRUE for NMPRG files only	FALSE
ISPROG	[BOOL]	TRUE for both PROG and NMPRG files	FALSE
ISEXECUTABLE	[BOOL]	TRUE for PROG, NMPRG, SL, XL files	FALSE
ISENCRYPTED	[BOOL]	TRUE for files encrypted by %SEC ENCRYPT	FALSE
ISQEDIT	[BOOL]	TRUE if the file code=111 (QEDIT)	FALSE
ISJUMBO	[BOOL]	TRUE if the file is a QEDIT "Jumbo" file	FALSE
ISPRIV	[BOOL]	TRUE if the file is PRIVileged	FALSE
ISDIR	[BOOL]	TRUE if a directory or account	FALSE
ISHIDDEN	[BOOL]	TRUE if file is not shown by :LISTF	FALSE
ISLINK	[BOOL]	TRUE if linked to another file	FALSE
ISBYTESTREAM	[BOOL]	TRUE if BYTE STREAM file	FALSE
ISXLSPoolFILE	[BOOL]	TRUE if NM SPOOL file	FALSE

```
%LISTF MYFILE.PUB.SYS,XL3
```

```
*****
```

```
FILE: MYFILE.PUB.SYS
```

```
FILE CODE : 0
```

```
BLK FACTOR: 1
```

```
REC SIZE: 256(BYTES)
```

```
BLK SIZE: 256(BYTES)
```

```
EXT SIZE: 0(SECT)
```

```
NUM REC: 0
```

```
NUM SEC: 0
```

```
NUM EXT: 0
```

```
MAX REC: 1023
```

```
FOPTIONS: BINARY, FIXED, NOCCTL, STD
```

```
CREATOR : MANAGER.VESOFT
```

```
LOCKWORD: **
```

```
GROUP ID: VESOFT
```

```
SECURITY--READ: ANY
```

```
WRITE: ANY
```

```
APPEND: ANY
```

```
LOCK: ANY
```

```
EXECUTE: ANY
```

```
**SECURITY IS ON
```

```
FLAGS : NO ACCESSORS
```

```

NUM LABELS: 0
MAX LABELS: 0
DISC DEV #: 2

SEC OFFSET: 0
UFID      : $055C0002 $4B8C11A6 $00593ABA $85033455 $1CBC9B07
VOLCLASS  : MPEXL_SYSTEM_VOLUME_SET:DISC

CREATED : SUN, AUG 11, 1991, 1:55 PM
MODIFIED: THU, DEC 24, 1992, 10:52 AM
ACCESSED: FRI, FEB 05, 1993, 8:15 AM
RESTORED: THU, DEC 30, 1993, 11:34 AM
STATE    : THU, DEC 24, 1992, 10:52 AM
LABEL ADDR: **

```

Attribute	Type	Returns	Example
FOPTIONS	[INT]	FOPTIONS of a file	\$2001
FMTFOPTIONS	[STR]	Formatted FOPTION value	"BINARY, FIXED, NOCCTL, STD"
CREATOR	[STR]	Creator name	"MANAGER"
CREATORACCOUNT	[STR]	Creator's logon account On pre-POSIX MPE/iX systems, this applied to native mode SPOOLFILES only.	"VESOFT"
OWNER	[STR]	Owner name	"MANAGER.VESOFT"
LOCKWORD	[STR]	Lockword (up to 8 chars, no blanks). Returns an empty string if the file has no lockword or the user does not have AM (for files within their account) or SM capability. (Remember, POSIX files do not have lockwords.)	""
FILEGROUP	[STR]	The file's GROUPID attribute	"VESOFT"
CREDATE	[DATE]	Creation date	8/11/91
CRETIME	[TIME]	Creation time	1:55 PM
CREDATETIME	[STR]	Creation date and time, a 12-character string formatted as "YYMMDDhhmmss", e.g. "900522050630" means 90/05/22, 5:06 AM and 30 seconds. Useful for comparing creation dates and times of files.	"900522050630"
ACCDATE	[DATE]	Last access date	2/5/93
ACCTIME	[TIME]	Last access time	8:15 AM
ACCDATETIME	[STR]	Last access date and time, "YYMMDDhhmmss" format.	"930205081530"
MODDATE	[DATE]	Last modify date	12/24/92
MODTIME	[TIME]	Last modify time	10:52 AM

Attribute	Type	Returns	Example
MODDATETIME	[STR]	Last modify date and time, "YMMDDhhmmss" format.	"921224105210"
RSTDATE	[DATE]	Last restore date	12/30/93
RSTTIME	[TIME]	Last restore time	11:34 AM
RSTDATETIME	[STR]	Last restore date and time, "YMMDDhhmmss" format.	"931230113422"
STATEDATE	[DATE]	State change date	12/24/92
STATETIME	[TIME]	State change time	10:52 AM
STATEDATETIME	[STR]	State change date and time, "YMMDDhhmmss" format.	"921224105210"

```
%LISTF MYFILE.PUB.SYS,4
-----FILE----- EXTENTS -BLKFACT- -----SECTORS-----  DEVICE
LABEL
NAME      CODE  NUM MAX  NOW BEST  USED NOW  SAVABLE  CLASS      LDEV
MYFILE                27  29   32  16      5930   1220   DISC      2
  Dev/Sector:    1/%000000242254  3/%000000242576  ...
```

Attribute	Type	Returns	Example
BESTBLOCKFACTOR	[INT]	Best blocking factor for this file	16
SAVABLESECTORS	[INT]	Total sectors savable by ALTFILE...;XLTRIM	1220
DEVICE	[STR]	Device class	"DISC"
FLABLDEV	[INT]	LDEV for the file label. Note: this isn't necessarily where the file resides.	2
LASTTEXTENTSIZE	[INT]	# of sectors in last extent	

```
%LISTF MYFILE.PUB.SYS,SEC
FILENAME  REC  TYPE  READ      APPEND  WRITE  LOCK  EXECUTE
MYFILE    80B  FA    AC        AL,GU  CR     CR     ANY
```

Attribute	Type	Returns	Example
ISRELEASED	[BOOL]	TRUE if file is :RELEASEd	FALSE
ISSECURED	[BOOL]	TRUE if file is :SECURED	TRUE

Attribute	Type	Returns	Example
HASACD	[BOOL]	TRUE if file has an ACD	FALSE
SECURITYR	[STR]	Allowed readers	"AC"
SECURITYA	[STR]	Allowed appenders	"AL, GU"
SECURITYW	[STR]	Allowed writers	"CR"
SECURITYL	[STR]	Allowed lockers	"CR"
SECURITYX	[STR]	Allowed executers	"ANY"
SECURITYAC	[STR]	Rights of users of class AC	"R"
SECURITYAL	[STR]	Rights of users of class AL	"A"
SECURITYGU	[STR]	Rights of users of class GU	"A"
SECURITYGL	[STR]	Rights of users of class GL	""
SECURITYCR	[STR]	Rights of users of class CR	"WL"
CSECURITY xxx	[STR]	Same as SECURITY xxx (where xxx can be R, A, W, ANY, AC, AL, etc.) except that it factors in the group and account security access matrices as well. Not available in VEFINFO.	
ISBOSS	[BOOL]	TRUE if executing user is SM or AM of file's account.	

Miscellaneous attributes:

Attribute	Type	Returns
ACDREQUIRED	[BOOL]	TRUE if ACDs are required.
VALIDCOLDLOADID	[BOOL]	(always returns TRUE on MPE/iX) .
EXCLUSIVE	[BOOL]	TRUE if being accessed exclusively.
FULLNAME	[STR]	Fully-qualified name (up to 26 chars, no blanks, no lockword). E.g, "CICAT.PUB.SYS"
FULLNAMELOCK	[STR]	Fully-qualified name (up to 35 chars, no blanks, with lockword). E.g. "GOD/BOG.PUB.VESOFT"
LABEEOF	[INT]	Total currently written user labels.
LABELFLIMIT	[INT]	Maximum number of user labels.
LOADED	[BOOL]	TRUE if loaded (:RUN or :ALLOCATED).
ID	[STR]	Text associated with this file.
IDCONTAINS(S)	[BOOL]	TRUE if the string S passed to this function appears in the file's identification string.

Attribute	Type	Returns
OPENEDSYSFAIL	[BOOL]	TRUE if a system failure occurred while the file was FOPENed and the file has not been FOPENed since then (useful for KSAM files to determine if KSAMUTIL >KEYINFO;RECOVER should be run against them). (Not available under MPE/iX).
PATH	[STR]	The PATH portion of a filename. For the file "/ROOTDIR/subdir/filename", the PATH is "/ROOTDIR/subdir/".
RESTORED	[BOOL]	TRUE if being :RESTORED. (not available under MPE/iX).
STORED	[BOOL]	TRUE if being :STORED. (not available under MPE/iX).
READING	[BOOL]	TRUE if opened for read access.
WRITING	[BOOL]	TRUE if opened for write access.

Functions:

Attribute	Type	Returns
ACCESSIBLE(<i>S</i>)	[BOOL]	TRUE if you — the running user — have the access specified by string <i>S</i> to the file, e.g. ACCESSIBLE("RWX") is TRUE if you have Read, Write, and eXecute access to the file.
ACCESSIBLEBY(<i>S1</i> , <i>S2</i>)	[BOOL]	TRUE if the file has ACDs and the user indicated by <i>S2</i> has the access specified by <i>S1</i> to the file. For instance, ACCESSIBLEBY("RW", "JACK.PROD") is TRUE if JACK.PROD has Read and Write access to the file. <i>S2</i> may also be "@.account" or "@.@". — ACCESSIBLEBY("X", "@.DEV") is TRUE if all users of DEV account has eXecute access to the file; ACCESSIBLEBY("R", "@.@") is TRUE if everybody has been granted Read access to the file. Note: At the moment, this only returns TRUE for files with ACDs; also, it only looks at what access the user has been explicitly allowed; it does not consider that the user may be allowed full access because he is the file's creator, AM or SM. This may be changed in future versions.
DEVICESECTORS(<i>I</i>)	[INT]	The number of sectors of the current file located on LDEV <i>I</i> , e.g. DEVICESECTORS(2).

Attribute	Type	Returns
<code>`MATCHES (S)</code>	[BOOL]	TRUE if filename matches fileset <i>S</i> , e.g. <code>MATCHES ("AP@.SOURCE.@")</code> .
<code>NEWERTHAN (S)</code>	[BOOL]	TRUE if the file has been modified later than the corresponding file in the targetfileset specified by the string <i>S</i> (see " Target Filesets ", above). Because MPE may change the modify date for program files when you merely RUN them, MPEX uses the create date rather than the modify date if the targetfile is a program file (PROG or NMPRG). If the targetfile doesn't exist, <code>NEWERTHAN (...)</code> returns TRUE.
<code>ONDEVICE (I)</code>	[BOOL]	TRUE if at least one extent of the current file is located on LDEV <i>I</i> , e.g. <code>ONDEVICE (1)</code> .

Text processing and string searching functions:

Attribute	Type	Returns
<code>FSEARCHSTRING (S)</code>	[INT]	Total number of lines containing the string <i>S</i> in the text of the current file: <pre>%LISTF @.SOURCE(FSEARCHSTRING('FOO')>0)</pre> finds all the files in the <code>SOURCE</code> group which have one or more lines containing the string 'FOO'.)
<code>FCONTAINS (S)</code>	[BOOL]	TRUE if the string <i>S</i> is in the file. For example: <pre>%LISTF @.SOURCE(FCONTAINS('FOO'))</pre> finds all the files in the <code>SOURCE</code> group which have one or more lines containing the string 'FOO'. This is equivalent to the example for <code>FSEARCHSTRING</code> , but you don't include ">0" as part of the selection criteria.)
<code>FSEARCHEXP (S)</code>	[INT]	Total number of lines that match the logical expression — just like the ones that can be passed to <code>%PRINT...;SEARCH=...</code> — specified by <i>S</i> . Remember, however, that since in this context <i>S</i> is a STRING, it must be enclosed in quotes: <pre>%LISTF @.SOURCE (FSEARCHEXP("'CUST-NO' or '& 'CUST-NAME'")>0)</pre> will find all files that have at least one line containing EITHER the string 'CUST-NO' or the string 'CUST-NAME'. In fact, <i>S</i> can include (in addition to the logical expression used by <code>;SEARCH=</code>) any of the MPEX <code>%PRINT</code> parameters (although not all of them are useful in this context).

Attribute	Type	Returns
		For example, you might say
		<pre>%LISTF @.FAIRY.TALES & (FSEARCHEXP("'happily';START=50;END=60")>0)</pre>
		which means "list all of the files in the FAIRY.TALES group that have the word 'happily' between lines 50 and 60.
		Perhaps the most useful keyword in this context is ;MAX=, which means stop reading the file after finding a given number of occurrences of the string.
		<pre>%LISTF @.TALL.TALES(FSEARCHEXP("CL'blue ox'; & MAX=2")=2), 3</pre>
		shows you the create/restore/modify/access dates (" ,3") of all the files in the TALL.TALES group that contain at least two occurrences of the string "blue ox" (upper- or lowercase), and stops searching each file after finding two occurrences of the string.

File attributes pertaining to IMAGE datasets

Note: These file attributes are relevant only for IMAGE datasets — if the current file is not an IMAGE dataset, these attributes will return unpredictable values.

```
%LISTF CUST01.VESOF.TLIST,DB
FILENAME SET NAME TYPE LDEV ENTRY CAPA- %FULL BLK SECTORS %BLOCK SECTRS
COUNT CITY FCT WASTED WASTED

CUST01 CUST M 2 13856 19009 72.9% 19 17024 0.4% 62
```

Attribute	Type	Returns	Example
DBSETNAME	[STR]	Dataset name	"CUST"
DBSETTYPE	[STR]	Dataset type ("M", "D", or "A")	"M"
DBSETENTRIES	[INT]	# of dataset entries	13856
DBSETCAPACITY	[INT]	Dataset capacity	19009
DBSETMAXCAPACITY	[INT]	Dataset max capacity	19009
DBSETFULLNESS	[REAL]	DBSETENTRIES//DBSETMAXCAPACITY	0.729
DBSETHIGHWATERMARK	[INT]	Highwatermark: highest record number ever used	
DBSETBLOCKFACTOR	[INT]	Dataset blocking factor.	19
DBSETENTRYLENGTH	[INT]	Dataset data entry length (in words).	

Attribute	Type	Returns	Example
DBSETNUMPATHS	[INT]	Number of paths linked to this dataset.	
DBSETMEDIALength	[INT]	Dataset "media record" length, i.e. length of each record including IMAGE pointers.	
DBSETBLOCKSIZE	[INT]	Actual size (in words) of each dataset block on disk.	
DBSETBLOCKWASTAGE	[REAL]	Fraction (from 0 to 1) of each dataset block that is wasted (because DBSETBLOCKSIZE is not a multiple of 128 words).	

File attributes pertaining to programs

Note: These file attributes are relevant only for program files — if the current file is not a program file, these attributes will return unpredictable values (usually 0 for integers, FALSE for booleans, and "" for strings).

All of these file attributes must be referred to as "PROG.*attrvar*". The "PROG." simply tells MPEX that this attribute pertains specifically to program files (just like "RFILE." tells MPEX — in those contexts, like %REPEAT...%FORFILES, where MPEX doesn't automatically assume it — that the attribute pertains to files in general).

Although "RFILE." can usually be omitted for general file attributes (except in %REPEAT...%FORFILES), "PROG." can not be omitted but must always be specified if you want to get program file information.

```
%LISTF DBUTIL.PUB.SYS,5
FILENAME  MAXDATA  STACK  DLSIZE  DBSIZE  #SEGS  CAPABILITIES
DBUTIL    31000   1200    0    3296    4  BA IA PM MR DS
```

Attribute	Type	Returns	Example
PROG.FMTCAPS	[STR]	Formatted capabilities	"BA IA PM"
PROG.PMCAP	[BOOL]	TRUE if program has PM capability	TRUE
PROG.MRCAP	[BOOL]	TRUE if program has MR capability	FALSE
PROG.DSCAP	[BOOL]	TRUE if program has DS capability	FALSE
PROG.PHCAP	[BOOL]	TRUE if program has PH capability	FALSE
PROG.IACAP	[BOOL]	TRUE if program has IA capability	TRUE
PROG.BACAP	[BOOL]	TRUE if program has BA capability	TRUE
PROG.HASPRIVSEG	[BOOL]	TRUE if program has any permanently privileged segments.	TRUE
PROG.ZERODB	[BOOL]	TRUE if CM program was :PREPped with ;ZERODB.	FALSE
PROG.NUMSEGS	[INT]	# of code segments in program.	4

Attribute	Type	Returns	Example
PROG.DBSIZE	[INT]	# of words in CM program's globald (DB) area.	3296
PROG.STACK	[INT]	CM Program's :PREP-time ;STACK=	1200
PROG.DL	[INT]	CM Program's :PREP-time ;DL=	0
PROG.MAXDATA	[INT]	CM Program's :PREP-time ;MAXDATA=	31000
PROG.ISOCTCOMPED	[BOOL]	TRUE if program was :OCTCOMPed	FALSE
PROG.NMSTACK	[INT]	NM Program's :LINK-time ;NMSTACK=	503808
PROG.NMHEAP	[INT]	NM Program's :LINK-time ;NMHEAP=	81920000

File attributes pertaining to SPOOL files

Note: These file attributes are relevant only for spool files, and should thus be used only on those commands that handle spool files (like %ALTSPoolFILE, %DELETESPoolFILE or %SHOWOUT). If you use them in connection with normal files, you'll get unpredictable information.

All of these file attributes must be referred to as "SPOOL.*attrvar*". The "SPOOL." simply tells MPEX that this attribute pertains specifically to spool files (just like "RFILE." tells MPEX — in those contexts, like %REPEAT...%FORFILES, where MPEX doesn't automatically assume it — that the attribute pertains to files in general).

Although "RFILE." can usually be omitted for general file attributes (except in %REPEAT...%FORFILES), "SPOOL." can not be omitted but must always be specified if you want to get spool file information.

```
:RUN SPOOK5.PUB.SYS
> SHOW 970
FILE #JOB FNAME STATE DEV/CL PR COP RFN OWNER
0970 #J350 $STDLIST READY LP 3 1 MANAGER.VESOFT
FILE LDEV LABEL SECTORS LINES TIME
0970 %2 %1173436 40 87 12:15 2/ 9/91
```

Attribute	Type	Returns	Example
SPOOL.ISACTIVE	[BOOL]	Spool file STATE is ACTIVE	FALSE
SPOOL.ISREADY	[BOOL]	Spool file STATE is READY	TRUE
SPOOL.ISOPENED	[BOOL]	Spool file STATE is OPENED	FALSE
SPOOL.ISLOCKED	[BOOL]	Spool file STATE is LOCKED	FALSE
SPOOL.ISDEFERRED	[BOOL]	Spool file STATE is DEFERRED	FALSE
SPOOL.ISSPSAVE	[BOOL]	Spool file has SPSAVE set	
SPOOL.OUTPRI	[INT]	Output priority	3

Attribute	Type	Returns	Example
SPPOOL.JOBTYPE	[INT]	1 if created by session, 2 if created by job, 3 if created by job and >INPUT from tape, 0 if created by session and >INPUT from tape	2 (job)
SPPOOL.JOBNUMBER	[INT]	Creating job number	350
SPPOOL.USER	[STR]	Creating user name	"MANAGER"
SPPOOL.ACCOUNT	[STR]	Creating account name	"VESOFT"
SPPOOL.JSNAME	[STR]	Creating job/session name.	
SPPOOL.FILE	[STR]	Spool filename	"\$STDLIST"
SPPOOL.SPOOLFILENUM	[INT]	Spool file number (#0xxx)	970
SPPOOL.HASFORMMSG	[BOOL]	TRUE if has forms message.	
SPPOOL.DEVICE	[STR]	Output device class. Note: Numeric devices (e.g. LDEV 6) will be returned as a string (e.g. "6").	"LP"
SPPOOL.JOBABORTED	[BOOL]	TRUE if job aborted as a result of an error in a command that was not preceded by "!CONTINUE".	
SPPOOL.NUMCOPIES	[INT]	Number of copies.	1
SPPOOL.NUMLINES	[INT]	Number of lines.	87
SPPOOL.READYDATE	[DATE]	Date the file was made ready	2/9/91
SPPOOL.READYTIME	[TIME]	Time the file was made ready	12:15

GROUP and ACCOUNT attributes of files

Suppose you want to find all program files that were :PREPped with ";CAP=PH", but that reside in groups (or accounts) that don't have PH capability. You can easily select `PROG.PHCAP`, but how do you select based on attributes of the group and/or account the file is in? Very easily! You just use "`DIRGROUP.attrvar`" and "`DIRACCOUNT.attrvar`" in the selection condition (or "`RFILE.DIRGROUP.attrvar`"/"`RFILE.DIRACCOUNT.attrvar`" in a %REPEAT...%FORFILES)!

In the above example, you would simply say

```
%LISTF @.@.(PROG.PHCAP and NOT DIRGROUP.CAP("PH"))
```

or

```
%LISTF @.@.(PROG.PHCAP and NOT DIRACCOUNT.CAP("PH"))
```

For a complete list of group and account attributes, see the MPE Group Attribute Variables and MPE Account Attribute Variables in the Object Attribute Variables section of the VEAUDIT User Manual.

Actually, DIRGROUP and DIRACCOUNT are really just shorthand for VEGROUPINFO and VEACCTINFO; e.g.

```
DIRGROUP.CAP("PH")
```

is the same as

```
VEGROUPINFO(RFILE.GROUP+"."+RFILE.ACCOUNT).CAP("PH")
```

See [Appendix B](#) for more information on these and other functions.

JOB/SESSION attributes

Note: Unlike previous attributes, which pertain to some type of file ("flat" files, databases, spoolfiles, etc.), or to MPE groups and accounts, these are attributes of jobs and/or sessions. They can only be used in commands and/or expressions that handle MPE jobs/sessions (like `! [RJOB.attrvar]` in a `%REPEAT...FORJOBS` construct, and the `VEJOBINFO(...).attrvar` function).

When we use "job" in the description of the following attribute variables, we really mean "job or session". However, some attribute variables are meaningful only for jobs (e.g. NUMCOPIES, ISSTDLISTDELETE, OUTPRI, etc.), while others make sense only for sessions (HPDTCPORTID, ISQUIET, etc.). If you try to use a JOB attribute against a SESSION, you may get an error.

Attribute variables of type string [STR] omit trailing blanks.

Attribute	Type	Returns
FMTLOGON	[STR]	Job logon, formatted as "jobname,user.account,group"
FMTJOB	[STR]	Job number, formatted as "#Snnn" or "#Jnnn"
JSNAME	[STR]	Job/session name
USER	[STR]	User name
GROUP	[STR]	Group name
ACCOUNT	[STR]	Account name
HOMEGRUJJP	[STR]	Home group name
INTROTIME	[TIME]	Time the job was introduced
INTRODATE	[DATE]	Date the job was introduced
LDEVIN	[STR]	Job input device
LDEVLIST	[STR]	Job output device
CURRENTCMD	[STR]	Last MPE command the job executed
INPRI	[INT]	Job input priority (15 = HIPRI)
OUTPRI	[INT]	Job \$STDLIST output priority
NUMCOPIES	[INT]	Job \$STDLIST number of copies

Attribute	Type	Returns
ISDEFERRED	[BOOL]	TRUE if the job is deferred (i.e. INPRI<JOBFENCE)
MAINPIN	[INT]	Job's main PIN
ISRESTARTABLE	[BOOL]	TRUE if the job is restartable
CPULIMIT	[INT]	Job's CPU time limit
ISEXEC	[BOOL]	TRUE if job is executing
ISSUSP	[BOOL]	TRUE if job is suspended
ISWAIT	[BOOL]	TRUE if job is waiting
ISINIR	[BOOL]	TRUE if job is being initialized
ISSCHED	[BOOL]	TRUE if job is scheduled
ISQUIET	[BOOL]	TRUE if user has issued a SET MSG=OFF command
STATE	[STR]	Job state ("EXEC", "SUSP", etc.)
LOCATTR	[INT]	Local attribute of user ID that job is logged on to
HPDTCPORTID	[STR]	The DTC PORT address/ID for the session
INSPoolFILENUM	[INT]	Job input spool file number
OUTSPoolFILENUM	[INT]	Job output spool file number
ISSTDLISTDELETE	[BOOL]	TRUE if job did a :SET STDLIST=DELETE
PRIVATE	[BOOL]	TRUE if the job's \$STDLIST spoolfile is PRIVATE
SPSAVE	[BOOL]	TRUE if the job's \$STDLIST spoolfile should be saved

SECURITY/3000 user profile attributes

In addition to all of the various MPE-related attributes described above, MPEX also gives SECURITY users access to various "user profile" attributes, like the user's real name, last logon date/time or even user-defined fields.

You can access these attributes using the `VEPROFILEINFO(profile).attr` and `VEPROFILEEXISTS(profile)` functions (see [Appendix B](#) for details).

For a complete listing of user profile attributes, see "User Profile Attribute Variables and Functions" in the Listing User Profiles section of the SECURITY User Manual.

SPECIAL MPEX TOPICS:

Using MPEX to compile programs

Say that you change your COPYLIB — now you have to recompile all of your many COBOL programs. (Even if you think that the COPYLIB module you changed affects only 10 programs, you may easily be mistaken, and if you are, a partial recompile may do more harm than good!) MPEX lets you do this in one command.

Although these examples show the COBOL compiler, MPEX has commands to compile using C, FORTRAN, PASCAL, RPG, and SPL as well. The format for each of these commands is virtually identical, so once you've learned how to use one compiler, learning the others is a fairly simple task. At the end of this section, specific differences between our commands and their MPE equivalents will be discussed.

What does the

```
%COBOL AP@.SOURCE,AP@.PUB,,,,MAXDATA=30000;CAP=PH
```

command do? Well, it takes all the files in the fileset "AP@.SOURCE" and compiles (and :PREPs) them into program files with the same names except that they reside in the PUB group (that's what the AP@.PUB indicates). Since the "*prepparms*" parameter was specified (MAXDATA=30000;CAP=PH), all the :PREPs will take place with those parameters.

This is one command that you might want to do in batch mode rather than online (otherwise it might tie up your terminal for quite a while). Saying

```
$$COBOL AP@.SOURCE,AP@.PUB,,,,MAXDATA=30000;CAP=PH
```

will execute all the compiles in a batch job stream (that's what the "\$" before the "COBOL" indicates). If there are any compile errors, the job stream will send you a message indicating this; in any case, the job stream will also send you a message when it's done. (Remember that the "\$" prefix can be used in front of any command to execute it in batch.)

Although the command is called %COBOL, it's actually more like a :COBOLPREP — the second parameter is the program file (not the USL file). If you want to specify the USL fileset, too, you can — it's the sixth parameter (after the "*newfileset*" and before the "*prepparms*"). If you only want to do compiles, not :PREPs, see [below](#).

GENERAL SYNTAX:

General syntax for CM compilers:

Syntax:

```
%CMcompiler textfileset
      [, [progfileset]
      [, [listfileset]
      [, [masterfileset]
      [, [newfileset]
      [, [uslfileset]
      [, [prepparms] ]]]]]
```

Compilers available: COBOL, COBOLII, COBOLIIX, FORTRAN, FTN, PASCAL, RPG, SPL

SPECIAL MPEX TOPICS:: Using MPEX to compile programs

General syntax for NM compilers (compile only, no link):

Syntax:

```
%NMcompiler textfilesset
      [, [objectfilesset]
      [, [listfilesset]
      [, [mastfilesset]                (COBXL only)
      [, [newfilesset]                (COBXL only)
      [, [libfilesset] ]]]]          (PASXL only)
      [;INFO= infostring]
      [;WKSP= workspacename]        (COBXL only)
      [;XDB= xdbfilename]           (COBXL only)
```

Compilers available: CCXL, COB74XL, COB85XL, FTNXL, PASXL, RPGXL

General syntax for NM compilers (with linking):

Syntax:

```
%NMcompilerLK textfilesset
      [, [progfilesset]
      [, [listfilesset]
      [, [mastfilesset]                (COBXL only)
      [, [newfilesset]                (COBXL only)
      [, [libfilesset] ]]]]          (PASXL only)
      [;INFO= infostring]
      [;WKSP= workspacename]        (COBXL only)
      [;XDB= xdbfilename]           (COBXL only)
```

Compilers available: CCXLLK, COB74XLK, COB85XLK, FTNXLLK, PASXLLK, RPGXLLK

In the above syntax diagrams, each of the filesets described **MUST** be entered in the order given or a "," should be included as a placeholder. This is true even when using the ;INFO=, ;WKSP= and ;XDB= parameters. If you do not specify all the files or filesets before any of these parameters, you must include placeholders as shown in these examples:

```
%RPG  GL###S, GL000P, , , , GL000U, MAXDATA=30000; CAP=PH; RL=XRL3
%COB85XL  MYSRC, , , , ;WKSP="MYWORK"
```

Also note that for the NM compilers, COBOL has two additional filesets as well as additional keywords while the PASCAL compiler has only one additional fileset.

Compiling many sources into one program

The "%COBOL AP@.SOURCE, AP@.PUB" command we talked about above compiled each source file in AP@.SOURCE into a corresponding program file in AP@.PUB.

What if you organize your system by having a single main body source file (e.g. the main menu) and a lot of \$CONTROL DYNAMIC subroutines? MPEX can handle this, too:

```
%COBOL  GL###S, GL000P, , , , GL000U, MAXDATA=30000; CAP=PH; RL=XRL3
```

This will compile all the sources in the fileset GL###S into the program file GL000P. Incidentally, it specifies that all the sources are to be compiled into the USL file GL000U; you

don't have to specify the USL file on this kind of compile, but you'd often want to (in case you want to later recompile just one of the modules).

Compiling without :PREPping

If you want to compile but not :PREP, you may just specify the USL fileset (or a single USL file) without a program fileset. For instance, saying

```
%COBOL AP@.SOURCE,,,,AP@U.PUB
```

will compile all the files in AP@.SOURCE into correspondingly-named USL files in the fileset AP@U.PUB (each USL file will have the same name as the source file but with a U at the end).

Similarly,

```
%COBOL GL###S,,,,GL000U
```

will compile all the GL###S files into the single USL file GL000U.

Compiling with a COPYLIB

To compile with a COPYLIB, you should simply issue an ordinary

```
:FILE COPYLIB=copylibfilename
```

file equation. If you're compiling in batch, e.g.

```
;%$COBOL GL###S,,,,GL000U
```

that's OK — just issue the :FILE COPYLIB= file equation online and MPEX will automatically copy it into the job stream.

Specific differences between MPE and MPEX compiler commands

In general, MPEX follows the same syntax as MPE (or MPE/iX) to compile programs. As the COBOL examples above show, however, certain portions of the command may be a FILESET instead of a single file name. Here are some other notable differences between our commands and MPE:

- The MPE/V compilers FTN and PASCAL do not support the "*master-file*" and "*new-file*" parameters supported by some of the other compilers (COBOL, FORTRAN, RPG, and SPL); however, they are included in these commands for consistency's sake — you should always skip them.
- When compiling using the MPE/iX compilers, you could specify the ";INFO=" string immediately after the last actual parameter given. i.e. if you compile a C program using an ";INFO=" string without specifying the object and list files, you could simply say

```
:CCXL MYSOURCE;INFO="compiler-directives"
```

However, MPEX requires you to insert placeholders in the command for any parameter not specified, so to do the above command in MPEX, you would need to type:

```
%CCXL MYSOURCE , , ;INFO="compiler-directives"
```

Using third party compilers with MPEX

The above commands and examples all deal with using compilers supplied by HP for many of their third generation languages (COBOL, FORTRAN, PASCAL, RPG), but what about fourth generation languages (such as TRANSACT) or compilers supplied by a third party? While we don't have specific commands for all of the compilers available, it is a simple matter to create a command file that uses %REPEAT... %FORFILES to compile several files with one command.

Here is an example command file that accepts two parameters — a source fileset and a target mask. All of the files in the source fileset are compiled using the generic compiler into the files specified by the target "mask".

```
PARM CS$SOURCESET
ANYPARM TARGETMASK
REPEAT
    FILE SOURCE=!MPEXCURRENTFILE
    FILE TARGET=! [OBJECTFILE(!TARGETMASK)]
    ECHO -----Compiling !mpexcurrenfile
    RUN ANY.GENERIC.COMPILER;INFO="*SOURCE,$NEWPASS";PARAM=12345

    (if this is a CM compiler)
        PREP $OLDPASS,*TARGET
    (if this is a native mode compiler)
        LINKEDIT $OLDPASS;TO=*TARGET

FORFILES !SOURCESET
```

As you can see, this is a very simple example and does not take into account such possibilities as compiling multiple files into a single program file or redirecting the output listings to a printer. This example presumes that the name(s) of the files to be used are passed as an ;INFO= parameter, but could just as easily be specific file equations for the source and object files.

If the compiler you are using prompts for the names of the files to use as the source, object, and even the resulting program, you can use this example:

```
PARM CS$SOURCESET
ANYPARM TARGETMASK
REPEAT
    ECHO -----Compiling !mpexcurrenfile
    RUN ANY.GENERIC.COMPILER&
        ;INPUT="!MPEXCURRENTFILE";INPUT="$NEWPASS"
    (PREP or LINK as appropriate)
FORFILES !SOURCESET
```

As you may have noticed, `MPEXCURRENTFILE` corresponds to the actual (fully qualified) filename of the sourcefile for each repetition through the loop, and the `OBJECTFILE(...)` function takes a parameter which it applies to the `CURRENTFILE` to create a `TARGET` file name.

MPEX HOOK

Executing MPEX and MPE commands, and UDCs from EDITOR, QUERY or any other utility

Let's say you're working in EDITOR. You write your program, compile it, find that there's a 'bug', make some changes, run it, make some more changes – the usual program development process. Unfortunately, with EDITOR, this will involve a lot of effort. You have to:

- Make some changes to your program
- Keep your program
- Exit EDITOR
- Compile your program
- Enter EDITOR
- Text your program
- Make some more changes
- Keep it again
- Exit EDITOR
- Run your program
- Enter EDITOR
- Text your program again
- Make some more changes...

Much of your time is spent in doing nothing more than texting and keeping your program, exiting and entering EDITOR. This is true not only of EDITOR, but also of QUERY, BASIC and many other programs. It would make sense to have any program be able to execute any MPE command — including compiles, RUNs, and UDCs.

We let you execute these commands, and many more, from any of these programs. For instance, you can say

```
:RUN EDITOR.PUB.VESOFT;LIB=P
/TEXT MYSRC
<< modify your source >>
/KEEP MYSRC
/:%:COBOL MYSRC,MYUSL
<< fix the 'bugs' the compile discovered >>
/KEEP MYSRC
/:%:COBOL MYSRC,MYUSL
/:%:PREP MYUSL,MYPROG
/:%:RUN MYPROG
<< make more modifications >>
```

```
/%Q                                << assuming Q is your QUERY UDC >>
<< run QUERY to examine your data base >>
>EXIT
<< exit from QUERY back into EDITOR! >>
/EXIT
:    << you're back in MPE >>
```

As you see, you've not only saved a lot of time for yourself but you've also saved a lot of computer resources by executing MPE commands directly from EDITOR!

The commands could just as easily have been MPEX commands — for instance, if your program aborts, you can, without exiting EDITOR, say

```
/%ALTFILE MYDATA; FLIMIT=FLIMIT+1000
```

All you need to do to use the new (HOOKed) version of EDITOR is to

```
:RUN EDITOR.PUB.VESOFT;LIB=P
```

Actually, MPEX HOOKed programs have a number of valuable features in addition to just being able to execute MPEX commands from them:

- A REDO facility is available that lets you redo up to the last 1000 commands that you've typed in the program (just like MPEX %REDO). This doesn't just mean "redo the MPEX commands you typed" — this includes all the commands you've entered in this particular execution of this program.
- For instance, say that you make an error in an 8-line QUERY >REPORT – you can redo the entire statement, make one modification, and have QUERY immediately re-execute all 8 commands.
- You can activate the MPEX HOOKed program's father process without terminating the MPEX HOOKed program itself. For instance, if from within MPEX (or any other process-handling environment) you type:

```
%QUERY.PUB.VESOFT;LIB=P
```

and then when in QUERY type

```
>%%
```

QUERY will be suspended and the father process (in this case, MPEX) will be re-activated. You could then execute whatever MPEX commands you like (including other %RUNs), and then, whenever you want to, type:

```
%QUERY.PUB.VESOFT;LIB=P
```

QUERY will be re-activated (which will be quite a bit faster than re-%RUNing it from scratch); your database will still be open, records that you did a >FIND on will still be found, etc.

- If you have SM capability, the program will be allowed to SAVE FILES IN OTHER ACCOUNTS. Thus, SM users of the MPEX HOOKed EDITOR can now /KEEP across account boundaries; SM users of the MPEX HOOKed FCOPY can specify ;TO=;NEW files in other accounts; and so on.
- Finally, you can (as we discussed earlier) execute arbitrary MPEX commands and MPE commands (including compile, %RUN, UDCs) from within any MPEX HOOKed program, e.g. EDITOR, QUERY, etc.

REDOing subsystem commands

Just like MPEX lets you save, %REDO, %LISTREDO, and %DO up to the last 1000 (or even more!) MPEX commands you've typed, so MPEX HOOKed programs can save, %REDO, %LISTREDO, and %DO up to the last 1000 commands you've typed inside them. This doesn't just mean that you can redo only MPEX commands you've typed — it means that you can redo ANY input you've entered. For instance, you might say

```
%QUERY.PUB.VESOFT;LIB=P
...
>FIND PRICE>100 and STATE<>"CA" and NUM-UNITS<10 and &
>>    SHIP-CODE="XX"
172 ENTRIES FOUND
>REPORT
>>D,CUST-NAME,30;D,PRICE,50;D,NUM-UNITS,70;end
>, ,
...
129) FIND PRICE>100 and STATE<>"CA" and NUM-UNITS<10 and &
130)    SHIP-CODE="XX"
131) REPORT
>>D,CUST-NAME,30;D,PRICE,50;D,NUM-UNITS,70;end
```

As you see, the saved command history includes all the input you've typed to QUERY — the FIND command, its continuation, the REPORT command, the various report sublines, etc.

All the features of MPEX's %LISTREDO, %REDO, and %DO are available in MPEX HOOKed programs except:

- Instead of typing just "LISTREDO", "REDO" or "DO", you must type ":LISTREDO", ":REDO" or ":DO" (note the ":"). This is done to avoid any confusion with subsystem commands. You can also use the ", , ", ", and ", ." abbreviations, without any ":" prefix. (In fact, if the program terminates when you enter a ":" — like SPOOK5 does — you'll HAVE to use the abbreviations.)
- If you want to save the command history in a permanent file, you should make a file equation for "*progrname*.REDO.VESOFT", where "*progrname*" is the filename of the MPEX HOOKed program. For instance, to save your EDITOR command history in the permanent file EDHIST, say

```
:FILE EDITOR.REDO.VESOFT=EDHIST
```

One other important thing that you should know is that all subsystem input is saved in the command history; also, all subsystem input is checked for a leading "%" or ", ". There's no

way for MPEX to know if EDITOR is prompting for a command or, say, an /ADD line; therefore, all input that you enter in /ADD mode will be saved in the command history together with all the EDITOR commands you typed at the "/" prompt.

Also, if you type

```
/ADD
 127.5   %ABC%
```

MPEX will see the "%" at the beginning of the input line and treat it is an MPEX command (even though the line was entered in /ADD mode); it'll try to execute the MPEX command "ABC%", which will probably fail. The same will happen if one of your input lines starts with a ",", ".
a ",", ".
One of the reasons we chose "%" and ",", as identifying characters for MPEX commands and redo commands is that it's unlikely you'll ever want to use them for your own purposes. If, however, you want to give the program true input that starts with a "%" or a ",", you need to prefix the input with a space.

One of the reasons we chose "%" and ",", as identifying characters for MPEX commands and redo commands is that it's unlikely you'll ever want to use them for your own purposes. If, however, you want to give the program true input that starts with a "%" or a ",", you need to prefix the input with a space.

Activating the father process

If in response to any prompt of an MPEX HOOKed program you type the two characters

```
%%
```

then the MPEX HOOK facility will re-activate the father process of this MPEX HOOKed program. For instance, if you run MPEX HOOKed EDITOR out of MPEX, you might say

```
:RUN MAIN.PUB.VESOFT
...
%RUN EDITOR.PUB.VESOFT;LIB=P
/TEXT MYFILE
/...
/%%
%      << MPEX prompts you for input >>
```

Now, if you say %SHOWTREE, you'll see

```
%SHOWTREE
25      C.I.
      123  MPEX.PUB.VESOFT (active) [you are here]
      195  EDITOR.PUB.VESOFT
```

EDITOR.PUB.VESOFT still exists as a son process of MPEX — if you now say

```
%RUN EDITOR.PUB.VESOFT;LIB=P
```

or

```
%RUN 195      << activate the PIN you saw in %SHOWTREE >>
```

or (if you have a UDC called ED that runs EDITOR.PUB.VESOFT),

```
%ED
```

you'll be brought back into EDITOR, at exactly the same point that you left it. Your file will still have been /TEXTed in, all the options you had set will still be set, and, of course, the re-activation of EDITOR would take a lot less time than re-running it from scratch would.

Of course, this will work equally well if you run a HOOKed program from any advanced process handler (like MPEX itself) and then type "%" in the HOOKed program. The purpose of the "%" command is to make it easy for you to switch from the HOOKed son process back to the father (which will later presumably be able to re-activate the HOOKed son).

If you do run a program directly from the MPE prompt and then do a % within it, the MPEX HOOK will terminate the program rather than suspend it – MPE doesn't let us suspend processes that are run from the Command Interpreter.

If you want to, you can say

```
:SETJCW VESOFTHOOKPPSONONLY=1
```

to indicate that "%" (Percent Percent) can only be typed from TRUE son process (i.e. not a process run directly from the CI). If you set this JCW and type "%" from a son of the CI, you'll get an error message; this might save you from accidentally terminating a process that you only wanted to suspend.

Building files in other accounts

One of the features of HOOKed programs is their ability to build files in other accounts (assuming, of course, that you have SM capability). For instance, if you're logged on as MANAGER.SYS and run EDITOR.PUB.VESOFT, you'll be able to say /KEEP MYFILE.DATA.AP thus saving your work file in another account. No need to log on into the AP account — no need even to /KEEP the file as some temporary file and then use MPEX's %COPY command to copy the file into another account — you can /KEEP files in other accounts directly. The same can be done with other HOOKed programs. If you say

```
:RUN FCOPY.PUB.VESOFT;LIB=P
>FROM=MYFILEEB.DATA.AP;TO=MYFILEAS.DATA.AP;EBCDICIN;SUBSET=100;NEW
>EXIT
```

then the HOOKed FCOPY will save the new file into the AP account (provided, of course, that you have SM capability).

The only thing you need to worry about are programs that are "too smart for their own good". Some programs actually check to see if you're trying to create a file in another account — if you are, they don't even try to FCLOSE the file (which is where the HOOK would intervene to allow the cross-account save), but immediately print an error message.

For instance, if you try to /KEEP across account boundaries in EDITOR and the /KEEP file already exists then EDITOR will print this error message:

```
*71*KEEP FILE MUST BE WITHIN LOG ON ACCOUNT"
```

In this case, you should

- `:PURGE` the `/KEEP` file first
- and then do the `/KEEP`.

If the `/KEEP` file DOESN'T already exist, EDITOR won't do the account check, and everything will be OK.

Most programs don't do any special cross-account check, so you can make them build files in other accounts with no difficulty:

- `FCOPY TO=;NEW` files, for instance, can be built across account boundaries (provided, of course, that you're using a HOOKed `FCOPY` and have SM).
- The contributed QUAD text editor can, when HOOKed and run with SM, `/KEEP` across account boundaries.
- The BASIC subsystem, if HOOKed and run with SM, can `>SAVE` programs across account boundaries.

MPEX HOOK preserves ACDs in EDITOR, TDP, etc.

If you're using Access Control Definitions (ACDs) on your files, you know that when you `/KEEP` a file in EDITOR, TDP or most other editors, the ACDs of the file you've kept over are lost. This is because a `/KEEP` purges the old file and rebuilds it; when the old file is purged, its ACDs are destroyed, and the new file is built without ACDs.

This can be a serious problem. The person who does the `/KEEP` will most likely forget to put the ACDs back on the file; even if he wants to, he might not know what the ACDs were because once the `/KEEP` is done he can no longer see them. (He'd have to remember to do a `:LISTF...`, `-2` before every single `/KEEP` he does, and an `:ALTSEC...;NEWACD=` — specifying every single pair! — afterwards.)

What is most likely to happen this way is that, very soon, all the files that you normally edit will lose their ACDs and all the additional security that ACDs give them.

Naturally, we wouldn't be telling you this horror story if we didn't have a solution up our sleeve. All HOOKed programs will automatically preserve the ACDs of any file that they purge and then re-build (which is exactly what a `/KEEP` does). To be precise, every time the program purges a file the HOOK remembers that file's ACD; if the next file the program saves has the same name as the just-purged file, the HOOK will impose that saved ACD on the new file.

Thus, if you do an EDITOR `/KEEP`, EDITOR will first purge the old file (at which point HOOK will save the old file's ACD) and then save the new file with the same name as the just-purged old file (at which point HOOK will put that saved ACD onto the new file).

Note that the ACD that's saved is the ACD of the file purged by the `/KEEP`, not the ACD of the file that was `/TEXT`d. If you say

```
/TEXT A
...
/KEEP B
OK TO PURGE B? YES
```

then the new file B will have the same ACD as the old file B, not the same ACD as A.

This is all done for you transparently — no need to specify any special keywords or options; whenever HOOK senses a purge followed by a save under the same name, it will retain the file's ACD.

If for some reason you don't want to preserve a particular file's ACD, you can always do an `:ALTSEC...;DELACD` of the file. If you feel that you NEVER want the HOOK to preserve ACDs for you, you should say

```
:SETJCW VESOFTHOOKNOKEEPACD=1
```

before running the HOOKed program. This will tell the HOOK to do nothing about ACDs, which would normally mean that the programs will be allowed to throw away ACDs at /KEEP time like they normally do.

Note that it is possible that some programs might not delete and rebuild files in the way that the HOOK expects. (For instance, if the program purges the old file using the `:PURGE` command rather than using `FCLOSE` disposition 4 or if the program builds the new file under a different name and then `:RENAMES` it.) In these cases, the HOOK might not be able to preserve the ACD because it won't really know exactly what the program is doing.

We've tested the "keep ACD" feature in EDITOR and TDP. Feel free to try it in other programs; however, it is possible that the HOOK might not be able to properly preserve ACDs for them.

How to %HOOK a program

Some programs are HOOKed by VESOFT at installation time (EDITOR, QUERY). To "HOOK" other programs, just execute the MPEX command

```
%HOOK progname
```

(To do this, you must have SM capability or be logged on into the VESOFT account.)

Whenever you get a new version of a HOOKed program — for example, a new version of EDITOR.PUB.SYS from HP — you should "re-HOOK" it. For example:

```
%HOOK EDITOR.PUB.SYS
%HOOK QUERY.PUB.SYS
```

Or, you might say

```
:RUN STREAMX.PUB.VESOFT
Enter filename: MPEXHOOK.JOB.VESOFT
Name of program to HOOK (if any)? EDITOR.PUB.SYS
Name of program to HOOK (if any)? QUERY.PUB.SYS
Name of program to HOOK (if any)? MYFILE.MYGROUP.MYACCT
Name of program to HOOK (if any)?
#J123
```

MPEXHOOK.JOB.VESOFT is a STREAMX job stream that can do the HOOKing for you.

How to run a %HOOKed program

To run "HOOKed" Compatibility Mode programs on MPE/iX systems, simply

```
:RUN program.PUB.VESOFT;LIB=P
```

To run "HOOKed" Native Mode programs on MPE/iX systems, you will also need to add the VESOFT procedures used by "HOOKed" programs to your system segmented library, SL.PUB.SYS. These procedures are kept in the USL file VEPROCU.PUB.VESOFT, and can be added to the system SL by streaming the job stream VEPROC.JOB.VESOFT. (It's a simple, non-privileged procedure, so there's no problem adding it to the system SL — just re-stream VEPROC.JOB.VESOFT whenever you do a system update.)

VEPROC also adds the VECMMND, VEOPEN and VECHECKLOGON procedures to the system SL.

See "What If Your Program Does a :STREAM Programmatically (With The COMMAND Intrinsic)?" in Batch Access Security, "Installing VEOPEN" in Database Security, and "Checking SECURITY Logons From Within Your Program" in Logon Security in the SECURITY User Manual for more details.

After adding the segments to SL.PUB.SYS, simply

```
:RUN program.PUB.VESOFT;LIB=P
```

or

```
:RUN program.PUB.VESOFT;XL="MPEXHKNL.PUB.VESOFT"
```

(the latter is especially useful if the program already has an XL list it must be run with; generally, MPEXHKNL.PUB.VESOFT should be added to the end of the list).

To make it easier to invoke the "HOOKed" programs, you may want to set the file MPEXUDC.PUB.VESOFT system-wide.

MPEX process handling

MPEX can be a powerful process-handling environment. Not only can you run programs from within MPEX, but you can also:

- Manage suspended programs, programs that suspend before re-activating MPEX instead of terminating. STREAMX is a good example of such program, as are all MPEX HOOKed programs (in which "%%" is a suspend command).
- Run son processes and pass them input — without having to build a special STDIN file!
- Press [BREAK] and resume MPEX rather than aborting the entire MPEX process tree.
- Use additional new MPEX :RUN command keywords, such as ;GOON and ;PRI=...

These features can be quite useful both in day-to-day work (e.g. you can easily jump from MPEX to EDITOR or QUERY without going through the overhead of constantly re-CREATEing each son process) and for various "MPEX programming" constructs.

Managing suspended son processes

Normally, when you run a program (from within MPEX, MPE or any other program) and then exit it, the son process (the program being run) dies. It closes all its files, forgets all its state information, and terminates. If you want to re-run the program, you have to re-run it from the beginning (which can take a good deal of computer resources).

Some programs are smarter than that. Take, for instance, SPOOK5. If you say (from MPEX):

```
%SPOOK5
SPOOK5 G.03.00 (C) HEWLETT-PACKARD CO., 1983
>...do some work...
> EXIT
```

you'll be back in MPEX as soon as you type the "> EXIT". HOWEVER, the SPOOK5 son process won't actually be dead — it'll only be SUSPENDED.

Now, when you again (from the same MPEX session) say

```
%SPOOK5
```

SPOOK5 will simply respond

```
>
```

As you see, SPOOK5 doesn't print a header because it's not really being re-:RUN. Rather, it's being re-activated, a much quicker and more resource-efficient operation. Your current spool file is still >TEXTed, your current line number is still unchanged, your current >MODE settings are the same as they were when you >EXITed SPOOK5 before. In fact, if we type

```
> EXIT
```

again in our suspended SPOOK5 son, we'll be back at MPEX. Now, we might say

```
%SHOWTREE
27      C.I.
      79      MPEX.PUB.VESOF (active) [you are here]
      119     SPOOK5.PUB.SYS
```

This shows you that your MPEX process has a suspended son; in fact, given the number on the left (the PIN, in our case 119 for SPOOK5), we can say

```
%RUN 119
>
```

to re-activate SPOOK5.

HOOKed programs

This can be particularly useful when you're running "HOOKed" programs, programs that have been modified (see the [MPEX HOOK](#) chapter) to honor MPEX commands, the multi-line REDO facility, etc.

SPECIAL MPEX TOPICS:: MPEX process handling

One of the things that every HOOKed program (e.g. EDITOR.PUB.VESOFT, QUERY.PUB.VESOFT, etc.) can do is suspend and re-activate the father process whenever you type a "%%". For instance, you might say

```
%RUN EDITOR.PUB.VESOFT;LIB=P
/TEXT MYFILE
...
/%%
```

The "%%" typed at the "/" prompt — essentially a new EDITOR command implemented for you by the HOOK — causes MPEX to be re-activated. If you do a %SHOWTREE now, you'll see:

```
%SHOWTREE
25      C.I.
    79      MPEX.PUB.VESOFT (active) [you are here]
    106      EDITOR.PUB.VESOFT
```

EDITOR.PUB.VESOFT still exists as a son process, with exactly the same /TEXTed file, current line pointer, /SET command settings, etc. Now, if you say

```
%RUN EDITOR.PUB.VESOFT;LIB=P
/
```

the EDITOR son process will be re-activated (in a fraction of a second), and you'll be back where you were when you typed the "%%".

The key here is that EDITOR (and QUERY and FCOPY and KSAMUTIL and so on) don't normally suspend — if you enter EXIT in any one of them, they'll terminate altogether. It's the MPEX HOOK that implements the new "%%" command which allows them to suspend instead of terminating.

A cautionary note

We think that the ability to manage suspended son processes can be very useful; however, it might sometimes surprise you (especially at first). For instance, if you say

```
%QUERY.PUB.VESOFT;LIB=P
>BASE=MYBASE
...
>FIND...
>%%
```

QUERY will still be alive, MYBASE will still be opened, and whatever locks QUERY may hold will still be held (which is, of course, what suspending-instead-of-terminating is all about). To terminate QUERY, you should say either

```
%QUERY.PUB.VESOFT;LIB=P
>EXIT
```

(since QUERY's >EXIT command does indeed terminate)

or

```
%KILL QUERY.PUB.VESOFT
```

MPEX's %KILL command (documented elsewhere in this manual) kills the son process whose name you specify.

Passing input to son processes

MPE's :RUN...;STDIN=... lets you run a program with its STDIN redirected to a disk file. But what if you don't want to build a disk file? What if you want to have a UDC such as:

```
ERASEDB !DBNAME
RUN DBUTIL.PUB.SYS,ERASE
```

that will erase a database? You want to take the database name from the !DBNAME UDC parameter, but DBUTIL.PUB.SYS,ERASE will prompt you for the database name. You could :RUN DBUTIL.PUB.SYS,ERASE;STDIN=..., but what will you use for the STDIN file? Surely you don't want to have one STDIN disk file for each database you might want to erase!

MPEX solves this problem by implementing a new %RUN command parameter called ;INPUT=. (Also, see commands %RUNCREATE, %RUNINPUT and %RUNACTIVATE). In the above example, you could simply say

```
ERASEDB !DBNAME
RUN DBUTIL.PUB.SYS,ERASE;INPUT="!DBNAME"
```

MPEX will take the ;INPUT=... parameter, write it to a special temporary file, and then RUN DBUTIL.PUB.SYS with STDIN redirected to that temporary file.

You can use the same construct for feeding multiple input lines — you have more than one ;INPUT= parameter on the same command, and each one will be written to the temporary STDIN file in turn.

For instance, a simplified version of MPEX's %EDIT command might have been implemented as follows:

```
PARM FILESET, CMD
REPEAT
  ECHO -----Editing !MPEXCURRENTFILE
  CONTINUE
  RUN EDITOR.PUB.SYS;INPUT="TEXT !MPEXCURRENTFILE";INPUT="!CMD";&
                                INPUT="KEEP";INPUT="EXIT"
FORFILES !FILESET
```

(see the %REPEAT...%FORFILES documentation for more details).

This runs EDITOR.PUB.SYS with ;STDIN=... redirected to a temporary file that contains the lines:

- "TEXT *filename*", where *filename* is the name of the current file;
- "*cmd*", where *cmd* is the EDITOR command being executed;

SPECIAL MPEX TOPICS:: MPEX process handling

- "KEEP", to keep the changed file back;
- "EXIT", to get out of EDITOR.

If we didn't have ;INPUT=, we really couldn't do this sort of thing, since the STDIN file to be passed to EDITOR changes from file to file.

One related feature that we provide is the ;INMSG keyword. This merely tells MPEX that the temporary file that contains the ;INPUT= lines is to be a message file rather than a standard-format file.

Why would you want to do this? Well, say that you want to have a son process that you keep suspended and "feed" commands to. You might run it with ;INPUT="xxx";INMSG, and then keep re-activating it, passing more ;INPUT="xxx" lines through this temporary message file.

A good example of this might be a UDC such as:

```
SPOOKCMD !CMD
RUN SPOOK5.PUB.SYS;INPUT="!CMD";INPUT="EXIT";INMSG
```

Now, if you say

```
%SPOOKCMD "TEXT #01234"
```

MPEX will create a SPOOK5 son process, pass it the command "TEXT #01234", and then pass it an >EXIT command which will re-activate MPEX. Now, you have a suspended SPOOK5 son process with a >TEXTed file; you may now say

```
%SPOOKCMD "L 1/20"
```

and the SPOOK5 son process will be activated and passed the "L 1/20" command — it'll list the first twenty lines of #01234 and then (because of the INPUT="EXIT") re-activate MPEX.

As you see, you can now execute SPOOK5 commands from within MPEX much like you could execute MPE commands from within EDITOR — just prefix the SPOOK5 command with "SPOOKCMD" and it'll be executed by the SPOOK5 son process.

:WAKE, MPEX's improvement to [BREAK] and :ABORT/:RESUME

One of the problems with process handling is [BREAK]ing and :ABORTing. If you press [BREAK] and do an :ABORT, the entire process tree will be killed (a phenomenon we call "mass processide").

If you run MPEX, run another program that gets into an infinite loop, press [BREAK], and do an :ABORT, everything will be killed — the program, MPEX, whatever other son processes MPEX might have, and all of their son processes.

What's the solution? Well, one solution is not to run programs that go into infinite loops...

Another, more practical alternative involves our WAKEUP program. To use it, you need to do two things:

- :SETCATALOG WAKEUDC.PUB.VESOFTE for your user, account (or even the entire system). This defines a UDC called :WAKE that can be used in place of :ABORT (more about that later).

- `%WAKEUP.PUB.VESOFT` in your MPEX session. We suggest that you put this into your `MPEXMGR`, `MPEXMGR.PUB`, or `MPEXMGR.PUB.VESOFT` file.

Now, say that you enter:

```
:RUN MAIN.PUB.VESOFT
...
%INFLOOP.TEST
<< program branches to never-never-land... >>
```

Now, instead of saying `[BREAK]` and `:ABORT`, you say

```
[BREAK]
:WAKE
%
```

The `:WAKE` command has awakened the MPEX process (that's why you get the "%"). The `INFLOOP.TEST` process is still around:

```
%SHOWTREE
25      C.I.
      79      MPEX.PUB.VESOFT (active) [you are here]
      106     EDITOR.PUB.VESOFT
      126     INFLOOP.TEST.DEV (active)
```

As you can see, `INFLOOP.TEST.DEV` is still active, chugging right along. You can either let it go (in which case it'll proceed just as if it were run with `;GOON` — see [below](#)) or say

```
%KILL INFLOOP.TEST.DEV
```

or

```
%KILL 126
```

The important thing to remember is that for `[BREAK]/:WAKE` to work, you must have done a `%WAKEUP.PUB.VESOFT` in your MPEX session (preferably in one of your `MPEXMGR` files) AND you must have a `:WAKE` UDC set for you. `WAKEUP` acts as a sort of "monitor" that's asleep, but can be awakened by the `:WAKE` command — when the `:WAKE` is done, `WAKEUP` is triggered, and it re-activates MPEX (which is suspended waiting for its infinite-loop son to stop).

Naturally, there's no need for the son to be actually in an infinite loop. It could just be taking a long time doing something you don't want to do — doing a `[BREAK]`, `:WAKE`, and then a `%KILL` can stop it.

The only thing you have to worry about is a son process that reads input from a terminal. When you do a `[BREAK]` and a `:WAKE`, MPEX is awakened, but the son process remains active — now, both MPEX and the son process are asking for terminal input at the same time. In this case, the first `%KILL` you type might very well be read by the son process rather than by MPEX. The son process will probably just reject this, and prompt for input again — keep entering `%KILLs` until one of them will be read by MPEX. (This is just the way MPE works when two or more processes have simultaneous reads pending on a terminal.)

%RUN...;PRI=...

The `;PRI=...` keyword of the `%RUN` command lets you indicate the priority with which you want the son process to run (CS, DS or ES). For instance, if you want to run some big CPU-hog report, you could say

```
%RUN OINKOINK;PRI=ES
```

The son process `OINKOINK` will now run in the ES queue, and you'll get that feeling of virtue for not slowing down other users too much.

%RUN...;STDIN=...;STDLIST=...

MPEX's `%RUN` command has `;STDIN=...` and `;STDLIST=...` keywords just like MPE's `:RUN` command does. However, MPEX's `;STDIN=...` and `;STDLIST=...` need not be just filenames — they can include an entire file equation (except for the word `:FILE`), e.g.

```
%RUN MYPROG;STDIN="MYMSG,OLDTEMP;REC=,,F;DEL";&  
      STDLIST="OUTFILE,OLD;SHR;GMULTI"
```

which is identical to

```
%FILE TEMPIN=MYMSG,OLDTEMP;REC=,,F;DEL  
%FILE PERMOUT=OUTFILE,OLD;SHR;GMULTI  
%RUN MYPROG;STDIN=*TEMPIN;STDLIST=*PERMOUT
```

The same thing, but with the extended `;STDIN=...` and `;STDLIST=...` keywords you can save yourself a file equation or two.

%RUN...;GOON

The `;GOON` keyword (which means "go on") merely says that the son process is to be run but MPEX should not be suspended. For instance, say that you have a big task (that requires no terminal input or output!) that you want to do in your session, but you don't want to lock up your terminal. You can say

```
%RUN BIGPROG;GOON
```

and have `BIGPROG` run as a son process while you keep doing whatever you want to be doing in MPEX. Watch out:

- If `BIGPROG` does terminal output, you might get its output interspersed with whatever other output you're getting from MPEX or another son process.
- If `BIGPROG` does terminal input, you might have it prompt for input at the same time as MPEX or another son process is prompting — this can be very confusing, since you won't know which program will actually get the input line you type.
- Don't confuse this keyword with the `%GOON` command prefix. Programs run using `:RUN...;GOON` are NOT reported by the `%SHOWGOON` command.

To be perfectly frank, the `;GOON` keyword is useful rather infrequently, mostly in sophisticated MPEX programming situations. If you do use it, you might want to also use the `;NOACTSONTERM` keyword (see the [%RUN command](#) documentation for more information).

`%RUN...;NOACTIVATE, ;NEW, and ;KILL`

These three `%RUN` command keywords govern how the son process is to be started and/or terminated:

- If you use `;NOACTIVATE`, the son process will be `CREATED` but not `ACTIVATED`.
- If you use `;NEW`, the son process will always be `CREATED`, even if a process with this name already exists.
- If you use `;KILL`, if the son process suspends instead of terminating, it will be `KILLED`.

These keywords are usually used for sophisticated MPEX programming applications. For instance, say that you want to have a UDC or command file that runs `SPOOK5`, executes a few commands, and gets back to MPEX. It might look something like:

```
SPOOKLIST !SPOOLFILENUM
RUN SPOOK5.PUB.SYS;INPUT="T !SPOOLFILENUM";INPUT="L ALL";INPUT="E"
```

— it'll run `SPOOK5`, `>TEXT` in the spool file you want, `>LIST` it, and then `>EXIT` back to MPEX.

However, what if you already have a `SPOOK5.PUB.SYS` son process? You certainly don't want to re-activate it, which is what the `%RUN` command would normally do — therefore, you say

```
SPOOKLIST !SPOOLFILENUM
RUN SPOOK5.PUB.SYS;NEW;INPUT="T !SPOOLFILENUM";INPUT="L
ALL";INPUT="E"
```

The `;NEW` keyword will guarantee that a new `SPOOK5` son process will be created. However, now the `SPOOK5` son is still alive, since the `>EXIT` command has only suspended it, not terminated it.

You might say

```
SPOOKLIST !SPOOLFILENUM
RUN SPOOK5.PUB.SYS;NEW;INPUT="TEXT !SPOOLFILENUM";&
INPUT="LIST ALL";INPUT="EXIT";KILL
```

to make sure that MPEX kills the son process after it's done. (You might also have said `INPUT="QUIT"` instead of `INPUT="EXIT"`, telling `SPOOK5` to execute the `>QUIT` command, which forces a termination.)

In most cases you will probably find it more convenient to use the `%RUNCREATE`, `%RUNINPUT`, and `%RUNACTIVATE` commands (documented earlier in this manual) rather than `;NOACTIVATE`, `;NEW`, and `;KILL`.

The MPEXPIN JCW

One other feature that can be useful for MPEX process handling applications: whenever MPEX runs a son that suspends, MPEX will set the JCW called "MPEXPIN" to the PIN of that son. You can then save that value in some other JCW, and then use it to re-activate or kill that son process, e.g.

```
%RUN SPOOK5.PUB.SYS;NOACTIVATE
%SETJCW SPOOKPIN=MPEXPIN
...
%RUN !SPOOKPIN
...
%KILL !SPOOKPIN
```

If you use MPEXPIN, you might also find the `SONALIVE(pin)` function to be handy — it returns true if the son process with the given PIN is still alive.

MPEX and your system security

Before we go any further, let us point out one of the fundamental principles of MPEX:

- **MPEX IN NO WAY DIMINISHES YOUR SYSTEM SECURITY.**

MPEX might make some things easier for people to do, but it'll never let anybody do anything to a file that they wouldn't be entitled to do without MPEX.

In fact, MPEX can greatly improve your system security by letting you easily %SECURE filesets, find released files, find privileged programs, and so on.

Removing "creator-only" restrictions

MPEX removes "creator-only" restrictions for the %RELEASE, %SECURE, %RENAME, and %ALTSEC commands.

In MPE, you can't :RELEASE, :SECURE, :RENAME or :ALTSEC a file unless you are its creator. You might have full access to it — be able to copy it and purge it — but not be able to :RENAME it. This struck us as a rather inconvenient feature, one that didn't add anything to your system security (why should, say, an account manager be prohibited from :RENAMEing a file in his own account). Therefore, our rule is different:

- Anybody who has READ, WRITE and EXECUTE access to a file can %RELEASE, %SECURE, %RENAME or %ALTSEC it.

This is, of course, in no way a security violation. Remember, somebody with READ, WRITE, and EXECUTE access to a file can already do whatever he please to it, from purging it to changing it to copying it. It only makes much more convenient something that somebody could always do the hard way.

Copying files into other accounts

Another thing that we allow is

- We let anybody with SM capability copy files into other accounts.

Standard MPE security forbids **ANYBODY** — even **MANAGER.SYS!** — from building files in other accounts. We think that this makes no sense; if you have SM, you could always log on to the target account, do the copy (possibly after **:RELEASE**ing the original file to make it accessible), and log back on to wherever you were before.

But why make the system manager do this? MPEX lets him **%COPY** files into other accounts (also, MPEX **HOOKed** programs allow SM users to build files in other accounts, so **HOOKed EDITOR** for instance allows cross-account **/KEEPS**). No security violation, but a great convenience!

Working on databases

MPEX lets you do several things to databases:

- **%COPY** them.
- **%RENAME** them.
- **%ALTFILE...;DEV=xxx** them (to move them from one disk to another).
- **%RELEASE** them.
- **%SECURE** them.
- **%ALTSEC** them.
- **%PURGE** them.

Naturally, file system security rules are not enough for checking database access — many databases are **>>RELEASEd** and count on their privileged file codes and **IMAGE** security to protect them.

The rule that MPEX uses is that:

- **ONLY**
 - the database's creator,
 - the account manager of the database's account, or
 - the system manager
 can execute any of the above commands against a database file.

Spool files

Normal MPE **:ALTSPoolFILE** and **:DELETESPoolFILE** are available only to the console operator and people who are **:ALLOWed** those commands. MPEX's **%ALTSPoolFILE** and **%DELETESPoolFILE** can be done:

- by the spool file's creator,
- by the account manager of the spool file,
- by the system manager, or
- by a user with **OP** capability.

Lockwords

In MPE, whenever you want to access a lockworded file, you must specify the lockword (even if you're SM or AM and can figure it out with `:LISTF...`, -1). This is no big deal on MPE's single-file commands, but say that you're doing a

```
%COPY @.@.AP, @.@.APNEW
```

— you certainly don't want to be prompted for the lockword of each lockworded file (or have the `%COPY` of that file fail if it's done in batch).

MPEX's rule on this is rather straightforward:

- If you have SM, MPEX will automatically supply lockwords on all lockworded files in the source fileset.
- If you have AM, MPEX will automatically supply lockwords on all lockworded files in the source fileset as long as the file is in your own account (files in other accounts won't have lockwords automatically supplied).
- If you specify a lockword on the source fileset, e.g.

```
%COPY @/FOO.@.AP, @.@.APNEW
```

then MPEX will automatically supply the lockword "FOO" on all lockworded files in the source fileset that have that lockword. In other words, non-lockworded files will be handled OK, and so will be those that have the lockword "FOO" — however, a file with any other lockword won't have it automatically supplied (and you'll be prompted for it when the time comes).

- If none of the above is true, the lockword will not be automatically supplied, which simply means that you'll be prompted for it just as you would in MPE (in batch, you'll get a lockword violation error).

This solution maximizes both ease-of-use and security — you don't lose any security since people with SM or AM can figure out the file lockword anyway (using `:LISTF...`, -1).

One thing that you must keep in mind with lockwords is that unless you explicitly specify a lockword in a target file (e.g. the second parameter on a `%COPY` or a `%RENAME`), the target file will be built without a lockword.

For instance, the

```
%COPY @.@.AP, @.@.APNEW
```

will build the `APNEW` files without lockwords. This is done for MPE compatibility — if in MPE you say

```
:RENAME X/LOCK,Y
```

or

```
:FCOPY FROM=X/LOCK; TO=Y; NEW
```

then the new file will be created without a lockword — therefore, the `%RENAME`, `%FCOPY`, and `%COPY` commands were made to act the same way.

What if you want to preserve the lockword? No problem! Just say

```
%COPY @.@.AP, @/=.@.AP
```

As you see, the target fileset (@/=.@.AP) was specified with a lockword of "=". The "=" wildcard (see "[Target Filesets](#)" in the MPEX Filesets chapter) tells MPEX to make that part of the target file be identical to the corresponding part of the source file; specifying an "=" in the lockword portion of a filename indicates that the target lockword should be identical to the source lockword.

Naturally, the same thing will work in %RENAME and %FCOPY commands.

GOD — gives SM capability and :ALLOWs all commands

Say that you are the system manager and have to log on as a normal user to do some troubleshooting; or, you need to log on under a particular user ID to run one of your programs. Whenever you log on as a non-SM user, many of the things that you are used to doing as SM no longer work. You can no longer access the files that you need to or run the programs that you need to; you can no longer :RUN some programs with the ;DEBUG parameter. The operating system looks at your logon ID, sees that this user doesn't have SM capability, and forbids you any SM-level operation. What MPE doesn't realize is that you (personally) really are the system manager — you just had to log on as a user who doesn't have SM capability.

VESOFT's MPEX product includes a program called GOD.PUB.VESOFT (which you should, of course, keep under a lockword) that temporarily grants you all the capabilities and all the :ALLOWs on the system. This way, if the system manager has to log on as a "plain vanilla" user, he can run GOD and be the system manager again, regardless of his logon. (The user ID's capabilities are not permanently changed — when you next log on as this user ID, it'll have the same capabilities that it did before; the system manager can also run a program called MORTAL.PUB.VESOFT to return to his pre-GOD state without having to log off.)

Since GOD has a lockword (whenever we install GOD on a system that didn't have it before, we actually assign it a random lockword!), only the person who knows the lockword (which should only be the system manager) can run it.

GOD can also be useful if some job stream that you have needs to acquire some special capability or :ALLOW — you can have it do a !RUN GOD.PUB.VESOFT, perform the operation that needs the capability or :ALLOW, and then !RUN MORTAL.PUB.VESOFT. Using STREAMX's \$NOPASS keyword, you can even do this without having to embed the GOD lockword into the job stream.

GOD is not just a great convenience for the system manager; it can actually ENHANCE your system security. If you don't use GOD, then you will inevitably have more :RELEASEd files, weaker group and account security, and more global :ALLOWs than necessary. Say that the system manager finds that he often has to do a :REPLY or a :STARTSPool when logged on as various user IDs — without GOD, he'll be quite likely to :ALLOW these commands globally since there would otherwise be no way for him to access these commands. If he has GOD, he can avoid :ALLOWing these commands because he will always have them available (if he knows the GOD lockword). Similarly, if the system manager finds himself needing various programs or data files when he logs on as non-SM users, he will be tempted to :RELEASE

SPECIAL MPEX TOPICS:: MORTAL — undoes a previous "GOD" operation

these files or otherwise keep them less secure than they should be. If he uses GOD, he will always be able to access any file he needs — again, only if he knows the GOD lockword.

If you have SECURITY you can protect it even further — see the SECURITY User Manual chapter on Restricting Access To GOD.PUB.VESOFT. In fact, SECURITY provides a number of useful features (like \$ALLOW, CHLOGON, and WITHCAPS) that eliminate the need for GOD.PUB.VESOFT in most circumstances — see the SECURITY User Manual for more details.

MORTAL — undoes a previous "GOD" operation

MORTAL is a program which sets the capabilities of the user who runs it back to their original state. This program is usually run after GOD is run to bring user's normal capabilities back (see "GOD" in this section of this manual):

```
:RUN MORTAL.PUB.VESOFT
```

MORTAL can also be used any time you need to change user's capabilities, while doing :ALTUSER; normally, the user must log back on to the system to have the new capabilities. Instead, he can run MORTAL to transfer the capabilities from the system tables to your session's capability area in the stack.

VEMODIFY — line editing made easier

VEMODIFY is a VESOFT-written procedure that provides a ONE-PASS, VISUAL FIDELITY line modification facility which is much easier to use than EDITOR's or TDP's /MODIFY (or the MPE :REDO command). It may be used in:

- An MPEX HOOKed EDITOR or TDP (using the /MV command).
- MPEX's %REDOs (if the HPREDOVEMODIFY JCW is set).
- :REDOs (or ', 's) in any MPEX HOOKed program.
- Your own EXPRESSION programs or command files by calling the VEMODIFY(...) function.

VEMODIFY's visual fidelity means that any modifications made to a line are echoed immediately rather than echoed after you press [RETURN]. As you do the modifications, VEMODIFY uses the power of your HP terminal to change the line as it appears on your screen at the same time as the line is changed in VEMODIFY's internal editing buffer.

The VISUAL REDO facility allows you to edit a line up to 150 characters in length. Since this is longer than a single screen line (even in 132 column mode), characters inserted in the first line will wrap-insert into the second line, and characters deleted from the first will pull back characters from the second line.

To use VEMODIFY in a HOOKed EDITOR, type

```
/MV linerange << e.g. /MV 1/4 >>
```

after :RUN EDITOR.PUB.VESOFT;LIB=P.

To use it in MPEX's (and HOOKed programs') REDO, say

```
:SETJCW HPREDOVEMODIFY=1
```

before entering the program.

VEMODIFY displays the line to be modified for you and prompts for modifications with the cursor right on top of the first character of the line. Then, when you type a printable character (e.g. 'B'), that character overwrites the one beneath it or is inserted before the one beneath it; when you type an unprintable character (e.g. [Control-B], space, etc.), VEMODIFY considers it a special command (see [below](#)).

Any unprintable control characters that were in the line when VEMODIFY was called are echoed as '.'s; no new unprintable control characters can be introduced into a line with VEMODIFY. If you make a mistake (e.g. type a control character that VEMODIFY does not consider a command, backspace beyond the first character in the line, etc.), VEMODIFY rejects your command and rings the terminal's bell.

The user should press [RETURN] only once — when the line modification is completed.

Callable VEMODIFY function for expression programs and command files

To use the VEMODIFY(...) function in your own expression program or command files, pass the string you want the user to modify to the function and set the result to a variable that contains the edited string. For example, suppose you want to "prompt" the user for some data, and you want to supply a default value based upon what the user typed in the last time. This is one way to accomplish this:

For a command file:

```
%ECHO Please enter the destination for the report
%SETVAR DEST VEMODIFY(SVAR('DEST','LP'))
```

For an expression program:

```
WRITELN ('Please enter the destination for the report');
DEST := VEMODIFY(SVAR('DEST','LP'));
```

This displays the prompt string ("Please enter the destination...") and then displays either the previous value of the variable DEST or "LP" if DEST has never been set. The user will then be in modify mode and can change the value for DEST as needed or simply press [RETURN] if all is well.

The VEMODIFY commands are:

Command	Mnemonic	Notes
[Control-A]	Append	Moves the cursor to the end of the line, thus allowing you to append text to your line. Same as [Control-L].
[Control-B]	Before	Enters Insert Character mode. All characters typed (until a [RETURN] or [Control-T]) will be inserted before the current character. Same as [Control-^].

Command	Mnemonic	Notes
[Control-C]	Case	Changes the case of the current character — upshifts it if it is lower case, downshifts it if it is UPPERCASE. Does not affect non-alphabetic characters.
[Control-D]	Delete	Deletes the current character. If at the end of line, deletes the line's last character.
[Control-E]	Erase	Erases everything from the current cursor position to the end of the line.
[Control-F]	Find	<p>Must be followed by another character.</p> <p>Finds the first occurrence of that character after the current cursor position, i.e. [Control-F] followed by an 'A' moves the cursor to the next 'A' in the line.</p> <p>[Control-F] [Control-W] finds the first character of the next word.</p> <p>If you type [Control-F] [Control-B] 'A', VEMODIFY will find the second occurrence of 'A' after the current character (B is the second letter of the alphabet); similarly, typing [Control-F] [Control-C] 'A' will find the third occurrence of 'A' after the current character, and so on.</p>
[Control-G]	Goof	Undoes all modifications made to this line.
[Control-L]	Lengthen	Same as [Control-A]. Useful when VEMODIFY is used on system console where [Control-A] is reserved.
[Control-N]	Number	Toggles displaying the line number of the current line. After this number has been displayed, pressing [Control-N] again will remove the line number.
[Control-O]	Overwrite	<p>Enters space overwrite mode.</p> <p>In this mode, typing a space causes the character beneath to be overwritten with a space rather than just causing the cursor to move forward. This mode is also initiated by typing any PRINTABLE character, and is stopped with [Control-T].</p>
[Control-Q]	Question	Displays this page of the manual, the "help" for VEMODIFY.
[Control-S]	Scan	<p>Same as [Control-F], but finds the first occurrence of the following character BEFORE the current cursor position.</p> <p>[Control-S] [Control-W] finds the first character of the previous word.</p>
[Control-T]	Terminate	<p>Terminates Character Insertion ([Control-B]) mode and Space Overwrite ([Control-O]) mode.</p> <p>[Control-T] [Control-D] cause the current line to be deleted.</p>

Command	Mnemonic	Notes
[Control-X]	Cancel	Redisplays the current state of this line and moves the cursor back to column 1.
[Control-^]	Insert	Same as [Control-B]. Useful when VEMODIFY is used on system console where [Control-B] is reserved.
" "	Space	Moves the cursor forward (default mode), blanks out the current character (Space Overwrite mode) or inserts a blank before the current character (Insert Character mode).
[Printable Character]		Overwrites the current character (Default mode) or is inserted before the current character (Insert Character mode). Turns on Space Overwrite mode.
[Backspace]		Moves the cursor back one column.
[Tab]		Same as [Control-I]. Moves the cursor forward 10 columns.

Online notepad

Has this ever happened to you?

You're on the phone with a friend or client, they mention in passing that their phone number has changed, and you suddenly discover that all of the pens on your desk have mysteriously and simultaneously run out of ink. After a bit of coaxing and lots of furious scribbling on what you thought was a bit of scrap paper (but turned out to be your last sheet of 20lb letterhead) you get the pen working only to realize you never heard the new number...

Use the ; prefix to enter a note

MPEX has a new "notepad" feature that is literally a single keystroke away! Any command line beginning with a semicolon (;) will be treated as a single line "note" to be entered in a notepad file! Beginning a command line with two semicolons will list or search the contents of the file.

So when Joe says, "Hey, I've just been moved to a new department over in the east wing of our complex, so starting next Thursday you can reach me at 555-1234", you simply type:


```
%;JOE:PHONE (work): 555-1234
```

and MPEX immediately files this bit of information away safely and cleanly (and you still have a sheet of letterhead available to write a letter congratulating your client on his promotion).

Everything you type after the notepad ";" character is written to a file along with a date and time stamp. When you list the file later, you can see when the notes were made. Since the time/date stamp takes about 20 characters to display, limit your notepad entries to 50-60 characters to avoid having the lines "wrap" when displayed.

Searching and printing notepad entries

The `;;` command is used to search and/or print Notepad entries:

<code>;;</code>	Prints the entire contents of your standard notepad: 
<code>;;text</code>	Without any wildcards: performs a caseless search for any lines containing <i>text</i> , equivalent to <code>%PRINT...;SEARCH=CL" text"</code>
<code>;;wildcardtext</code>	With (<code>@</code> , <code>?</code> , <code>[...]</code>) wildcards: performs a match against <i>wildcardtext</i> , equivalent to <code>%PRINT...;SEARCH=R MATCHES "text"</code>
<code>;;.aux-parms</code>	Append supplied <i>aux-parms</i> to the internal <code>%PRINT</code> command, equivalent to <code>%PRINT...;SEARCH=aux-parms</code> Can be used to specify search expression (e.g. <code>;;."text"</code>) and/or supply additional <code>%PRINT</code> command parameters (e.g. <code>;;. ;OUT=\$STDLIST</code>).

Since we actually use our `%PRINT` command internally, you can use the `;;.` syntax to specify a search string and/or regular MPEX `%PRINT` parameters. For instance, to print the entire file to a printer, use `;;. ;OUT=*LP`. To print the file without stopping (on screen), use `;;. ;PAGE=0`. As these are translated to `;SEARCH=;OUT=*LP` and `;SEARCH=;PAGE=0`, (the search argument is a NULL) the `;SEARCH=` keyword is effectively ignored.

By default, we apply the `;NUM` parameter to the `%PRINT` command in order to display line numbers (for editing, see next section), if this causes lines to "wrap around" your display, then you can append `;UNN` to the `;;.` command to avoid this problem.

Editing or deleting notepad entries

It should be fairly obvious that this feature is not intended to replace a full-scale editor for maintaining a simple text file — there are plenty of editors already on the market available for such tasks. Instead, this is intended to be a simple, quick, extension to MPEX that allows for rapid entry and retrieval of simple notes when it isn't convenient to grab a piece of paper or run a different program. However, just offering the ability to add and subsequently find notes is not sufficient. After all, information changes over time, records need to be updated, and eventually the information become obsolete and should be deleted. Therefore, we also provide the ability to modify lines and delete lines that are no longer needed.

Earlier we mentioned that listing the contents of a notepad with `;;` would, by default, be numbered for "editing". Entering `;M[num]` will modify the specified line number and `;D[num]` will delete the specified note line. The default for either of these options is the last line in the notepad.

<code>;M[num]</code>	Modifies last (or specified line <i>num</i>) entry. Modifications are limited to the data portion of a note, not the timestamp.
<code>;D[num]</code>	Deleted last (or specified line <i>num</i>) entry.

The modify command makes use of our visual-redo facility if you have our JCW set for this option, otherwise it uses the normal editor / command interpreter editing commands.

At the moment, this is the extent of the editing capabilities available — modifying incorrect lines (for instance, to correct the spelling of a particular item or update the status of a project) and deleting lines no longer deemed necessary (presumably, for tasks that have been completed). If you find that you must edit the file a bit more extensively, say for instance to add a paragraph or two to a note, then you should use a regular editor to edit the appropriate file (see the section [below](#) on how the files are named). One thing to be aware of when editing a note file manually is that the date and time stamp area of the file (the first 17 characters) cannot be "edited" using the ";M" command.

Multiple notepads — a mini-file cabinet

You actually have access to 11 separate notepads — your default notepad and ten others numbered from 0 to 9. If you precede the notepad character with a number from 0 to 9, then the note will be written to the corresponding notepad. (of course, "#;" will list the contents of that particular notepad).

Notepad file names – naming using VESOFTNOTEFILE variable

How do we determine the name of your notepad? If you did not log on with a session name, i.e. you logged on simply as `CLERK.OFFICE,MYGROUP`, then we use a file called `VENOTE.MYGROUP.OFFICE`. (actually, it is "VENOTE?" where the "?" character is replaced by the notepad number or blank for your default notepad). If you logged on with a session name, as in `"BOB,MGR.DEV"`, then we use a file called `"BOB?.VENOTE.VESOFT"`. Since we append a "?" character to the end of your session name, and MPE limits file names to 8 characters, we only use the first 7 characters of your session name so that we don't generate an error trying to create a 9 character name.

Using your session name is great if every session name is unique (since you can refer to the same notepad files regardless of where you log on) or if everyone logs on without a session name but in different groups (in which case the file `VENOTE?` is built in each group), but what if everyone logs on as `"CLERK.OFFICE,PUB"` or you have two users who log on with the session name of `"JOE"`? In these cases, you should set a special variable, `VESOFTNOTEFILE`, to a filename of your own choosing and MPEX will use that file for keeping notes. Note that in order to use the multiple notepad feature, you must include a question mark as part of the filename.

Changing the notepad prefix character (;)

You can change what character MPEX recognizes as the "notepad character" by setting the special variable `MPEXPREFIXNOTE` to either the ASCII value of a character or the actual character itself. In fact, you may be forced to set this variable if you use an `MPEXPREFIXDEF` variable for the ";" character since the `MPEXPREFIXDEF` character will override the notepad action. See the section of our manual regarding [SPOONFEEDing](#) for details on the `MPEXPREFIXDEF` variables.

MPEX COMMAND REFERENCE

MPEX recognizes many different commands. Some of them have no analog in MPE; others are quite similar to normal MPE commands. In general, whenever MPEX prompts you (with a "%") for a command, you can type one of the following:

- An ordinary MPE command, including commands such as %RUN, %EDITOR, %SHOWCATALOG, etc.
- A UDC invocation. (Normally the **COMMAND** intrinsic doesn't let you execute UDCs — we're rather proud to say that we were the first product [outside HP's Command Interpreter] that let you execute UDCs from within it.)
- One of the special MPEX commands (listed below), such as %COBOL, %COPY, %INPUT, %SETVAR, etc.
- One of the special MPEX commands that can handle filesets (%COBOL, %COPY, %EDIT, etc.), prefixed by a "?" — this tells MPEX to prompt you Yes/No for each file in the fileset and execute the command only on those files for which you say "Yes".
- Any command (MPE or MPEX) prefixed with a "\$", indicating that the command is to be executed offline (in a batch job stream of its own).
- The name of a program file (fully-qualified or not), which is then automatically :RUN; for instance, saying

```
%DBUTIL
```

will automatically do a :RUN DBUTIL.PUB.SYS even though you don't have such a UDC!

- The name of a command file, an ordinary EDITOR file that contains a set of commands to be executed. This is very similar to a UDC, but it doesn't have to be :SETCATALOGed — just entering its filename causes it to be executed.

How does MPEX decide which one of these to do? Suppose you have both a program and a UDC named FIBAR. Should MPEX execute the UDC or run the program?

Basically, MPEX follows the same rules as MPE/iX:

- first look for a UDC;
- next, see if the command is an MPEX command; if not,
- see if the command is an MPE command;
- finally, if none of the above is true, look for a program file or a command file matching the command name, execute that.

There are several things that can change this order:

- if you precede the command with %XEQ, MPEX will look for a command file or program file, and if it doesn't find one, print an error;
- if you precede the command with %RUN, MPEX will look for a program file, and if it doesn't find one, print an error;
- if you precede the command with a ":", MPEX will not look for an MPEX command (but it will emulate MPE/iX commands on MPE/V systems);
- if you precede the command with a "!", MPEX will not look for a UDC — use this if you have a UDC with the same name as an MPEX command. You can precede "!" with a "\$" to execute the command "offline" (see "[Batch execution](#)" in this manual).

Just like in MPE, any MPEX command can be continued with an "&". For compatibility with MPE/V and older versions of MPEX, MPEX inserts a space character in the command line in the position where the "&" was and includes leading spaces on the continued line. MPE/iX, however, does not include a space for the "&" and strips leading spaces from the continued line. If you want to tell MPEX to use the MPE/iX behavior, you should set the variable VESOFTCONTINUENOSPACE to 1 in your MPEXMGR.PUB.VESOFT file.

```
%SETVAR VESOFTCONTINUENOSPACE 1
```

In addition (also like MPE), the length of the entire command is limited to 279 characters, after all variables and expressions within the command have been "resolved" (see ["Using MPEX variables"](#), in this manual).

The following are all examples of valid MPEX commands:

```
%SHOWJOB JOB=@J
```

which executes the ordinary MPE command :SHOWJOB JOB=@J.

```
%RUN MYPROG;LIB=G;PARAM=17
```

which executes the MPE command :RUN MYPROG;LIB=G;PARAM=17. Although such a command can't normally be executed through the COMMAND intrinsic, MPEX can execute it nonetheless.

```
%S J
```

which, if you have a UDC called S, will invoke it with parameter "J".

```
%DELETESPOOLFILE $STDLIST. @.(SPOOL. OUTPRI<=3)
```

which will delete all the \$STDLISTs in the system that have an output priority of 3 or less (for more information, see the [%DELETESPOOLFILE](#) command description in this manual).

```
%%$COBOL AP@.SOURCE, AP@.PUB
%%$RUN MYPROG;LIB=G;PARAM=17
```

which will execute the specified commands in batch jobs (one batch job for the COBOL command and one for the RUN command). As you see, MPEX allows easy offline execution of either MPEX commands or normal MPE commands – just prefix them with a "\$".

```
%DBUTIL
```

which will run the program DBUTIL.PUB.SYS. Actually, it will (by default)

- first try to run DBUTIL (in your logon group);
- if it can't find DBUTIL, it'll try to run DBUTIL.PUB (in your logon account);
- if it can't find DBUTIL or DBUTIL.PUB, it'll try to run DBUTIL.PUB.SYS.

Thus, if you had a program called MYPROG in your own group and account, you could run it by simply saying

```
%MYPROG
%FROBOZZ 10, 20, 30
```

which will execute the commands in the file FROBOZZ (or FROBOZZ.PUB or FROBOZZ.PUB.SYS), passing to them as parameters 10, 20, and 30. The file FROBOZZ might look like:

```
PARM WIDGETS, GADGETS=33, FRAMASTATS, THINGAMABOBS=55
BUILD XXFILE;TEMP;REC=! [WIDGETS+GADGETS+FRAMASTATS]
RUN TWIDDLE;PARM=!WIDGETS;INFO="G!GADGETS"
RUN TWADDLE;PARM=!FRAMASTATS;INFO="XXX!THINGAMABOBS"
PURGE XXFILE,TEMP
```

— just like a UDC, except that instead of the header line "FROBOZZ WIDGETS, ...", the header says "PARM WIDGETS, ...".

%ABORTJOB

Syntax:

```
%ABORTJOB userset
           [;NOVERIFY]
           [;QUIET]
```

Examples:

```
%ABORTJOB @.AP-MGR.AP-RON,@.@
%ABORTJOB #J1234+#J1235
%ABORTJOB ONLINE-LDEV=!HPLDEVIN-LDEV=!HPCONSOLE
%ABORTJOB QREPORT,@.@
%ABORTJOB ACCESSING=CUST.DB.PROD
%ABORTJOB #A10 (aborts a STREAMX scheduled job)
%ABORTJOB @J&SCHED (aborts only MPE-scheduled jobs)
%ABORTJOB SCHED-@A
```

The %ABORTJOB command is an enhanced version of MPE's :ABORTJOB command. It allows you to specify which jobs to abort using VESOFT usersets (see the Usersets section in the Logon Security chapter of the SECURITY User Manual).

By default, ABORTJOB will show the job numbers and logon IDs of all the jobs that are to be aborted and then ask you if it's OK to abort them. If you do not reply "Y", the jobs will not be aborted. This can help you avoid accidentally aborting many jobs that you don't actually want to abort.

If you want to do the %ABORTJOB without prompting for verifications, just append ";NOVERIFY" to the command, e.g.

```
%ABORTJOB BATCH&EXEC;NOVERIFY
```

This will still show the numbers and logon IDs of the jobs that are being processed.

If you used the ";QUIET" keyword instead, e.g.

```
%ABORTJOB @.AP&WAIT;QUIET
```

then both the verification request and the job numbers/logon IDs will not be displayed.

For compatibility with MPE, if you specify a single job/session number (e.g. "%ABORTJOB #J1234") you will not be asked for verification, and the job number/logon will not be displayed, just as if you had said ";QUIET".

ABORTJOB and STREAMX scheduled jobs

Our STREAMX product (part of SECURITY/3000) has the ability to schedule jobs based upon a date, time or event, with an option to repeat as needed. These jobs are shown by our %SEC SHOWSCHED command with a "#A" that specifies each scheduled job. For example, entering %ABORTJOB #A12 would abort scheduled job #12 in our job scheduler. Once a job is scheduled, it keeps the same #A number until it is launched or aborted or in other words, the %ABORTJOB command doesn't change scheduled job numbers.

The %ABORTJOB command also supports our %SHOWJOB usersets of @J, @S, and @A.

See the chapter on repeating jobs in the STREAMX section of the SECURITY manual for more details.

%ALARM, %SHOWALARM, %DELETEALARM

Syntax:

```
%ALARM ["{time|booleanexpression"}"], [message]
        [;TO= userSet] | [;LOGON] | [;TERMINAL] | [;SESSION]
        [;NOHP]
        [;scheduleparms]
        [;CHECKEVERY= interval]
        [;REPEAT={DAILY|WEEKDAYS| list of days}]
        [;WHEN= expression]
        [;WHENEVER= expression]

%SHOWALARM
        [;ALL]
        [;FROM= mpeuserSet]

%DELETEALARM alarmnumber
```

Examples:

```
%ALARM "11:45AM", "Almost time for lunch!"
%ALARM "2:30PM", "Don't forget our meeting at 3:00";TO=@.DEV
%ALARM 'JSCOUNT("REPORT1,BATCH.PROD")=0', "REPORT1 is done!"
%ALARM 'JSCOUNT("BILL,@.@")>0', "Bill has logged on!"; &
CHECKEVERY=5
%ALARM "HPJOBCOUNT>3", "More than 3 jobs are logged on!"; &
TO=LDEV=20
```

```
%ALARM , "FORBIN Project is due TODAY!"; TO=TERMINAL; &
    DATE=11/05/91; AT=9:00

%SHOWALARM
%SHOWALARM;ALL
%SHOWALARM;FROM=@.DEV

%DELETEALARM 1
```

The %ALARM command uses the BACKG background job (see the [%BACKG command](#) in this manual for details) to send the specified message at some given future time. Alarms can be set by specifying:

- the time-of-day;
- a boolean expression (the message will be sent when the expression becomes TRUE);

MPE :STREAM-type scheduling parms (";AT=", ";DAY=", ";DATE=", or ";IN=").

Note that these are entered as separate parameters and not within the first parameter.

By default the message will be sent to your logon (i.e. to the "*session,user.account,group*" you were logged on as when you set the alarm); however, you can specify that the message be sent to any VESoft userset (see "Usersets" in the SECURITY User Manual for details on usersets) using the ;TO=*userset* option.

For example, to send the message to all users logged on with SM capability, you might say something like

```
%ALARM 'JSCOUNT("LDEV=20")>0', Someone logged on to the console; &
    TO=CAP=SM
```

Three shorthand keywords are also provided:

- ;LOGON (the default) means the same as ;TO=!HPJOBNAME,!HPUSER,!HPACCOUNT,!HPGROUP (i.e. notify anyone logged on exactly the same way as the person who set the alarm).
- ;TERMINAL means the same as TO=LDEV=!HPLDEVIN (i.e. notify anyone logged on to the terminal the alarm was set from, useful if you frequently log off and back on with a different user ID, but at the same terminal).
- ;SESSION means the same as ;TO=#!HPJOBTYPE!HPJOBNUM (i.e. notify the particular session that set the alarm, not all users that logged on the same as the user who set the alarm. This is useful if you log on with a user ID that is the same as other users, but want to limit who sees the alarm).

If the message (or the boolean expression) contains embedded commas, semicolons or quote characters, enclose the entire message (or boolean expression) in quotes.

Normally the message is sent using the equivalent of our %WARNF command (documented in this manual), which writes the message over the function key labels on HP terminals. If you know that the terminal to which the message will be sent is NOT an HP terminal (or emulator), you might want to use the ;NOHP option so that we won't send the escape sequence to write over the function keys.

Unlike the `:TELL` (or `%TELL`) command, `%ALARM` messages do not include any information about who set the `%ALARM`. To avoid having to manually include your name (or session number) in every `%ALARM` message you set, MPEX will prefix the message with the contents of the variable `MPEXALARMPREFIX` (if you `%SETVAR` it).

For example, to prefix all `%ALARM` messages with "From Ron: ", you would do a

```
%SETVAR MPEXALARMPREFIX "From Ron: "
```

before you do any `%ALARMS`. In fact, you may want to include this `%SETVAR` in your `MPEXMGR` file (see "[Specifying commands to be executed each time MPEX is entered](#)" in the Running MPEX section of this manual).

Repetitive alarms

The `%ALARM` task accepts the same repetitive keywords as `STREAMX` for scheduling repeating alarms. The `;REPEAT=...` keyword lets you use the values `DAILY` for every day of the week, `WEEKDAY` for every "work" day (excludes Saturday and Sunday), a specific day of the week or a list of days. For specific days, you need to supply at least the first three letters of each day you want the alarm to be sounded.

Ordinarily, when you use the `;REPEAT=...` keyword, you would also use `;AT=` to specify a particular time each day that the alarm should be triggered. If you use `;REPEAT=...` without `;AT=...`, the scheduled time for the alarm will be the time of day that you issued the command. So if you type:

```
%ALARM "","Quitting Time!";REPEAT=WEEKDAY
```

at 4:00 in the afternoon, then this alarm will be executed at 4:00 PM each weekday.

The keywords `;WHEN` and `;WHENEVER` are used to specify a condition that when true will signal an alarm. `;WHEN` is used to set up a single event alarm — when the condition becomes `TRUE`, the alarm is triggered and deleted from the scheduler. `;WHENEVER` is used to set up a multiple event alarm — as long as the condition is `TRUE`, the alarm will be sounded.

Security considerations

For security reasons, non-SM users are not allowed to use certain functions in the boolean expression when they set an alarm (some functions are not even allowed for SM users, because their use might hang the `ALARM` background process, but most of these wouldn't be very useful in an alarm anyway). The restricted functions include:

- the `MPE(...)`, `PAUSE(...)`, and `PRINTOPREPLY(...)` functions;
- all `VEFaccess` and `VEDBaccess` functions (like `VEFOPEN`, `VEDBOPEN`, etc.);
- the `LOCKWORD`, `ACCESSIBLEBY`, `FSEARCHEXP`, and `FSEARCHSTRING` file attributes (for example, using the `VEFINFO` function);
- all of the terminal input functions (`INPUT`, `READSTRING`, etc.)
- all of the variable-setting functions (`SETVAR`, `SVARSET`, etc.)
- all of the object info functions (`VEUSERINFO`, `VEACCTINFO`, etc.)
- all `SECURITY`-related functions (`SECURITYLOG`, `VEPROFILEINFO`, etc.)

Performance considerations

By default, the ALARM task wakes up and checks each alarm once each minute. This can be changed by setting the variable `VESOFTALARMPAUSE` in the `BACKG` initialization file. For example, adding the line:

```
SETVAR VESOFTALARMPAUSE 15
```

to the file `BACKGINI.DATA` will cause the ALARM task to wake up every 15 seconds or 4 times each minute.

Each time the ALARM background task wakes up, every alarm that has been set will be tested. If you set an %ALARM based on an expression that uses a lot of system resources (CPU, disk I/Os, etc.) to evaluate, you may want to check that particular alarm less frequently. When you enter an %ALARM command, simply append the `;CHECKEVERY=` keyword to tell the ALARM background task how many intervals to skip before checking the particular expression you are entering.

For example:

```
%ALARM "JSCOUNT('COMPILE,@.&&EXEC')=0", Compiles are done!; &  
CHECKEVERY=10
```

This means "notify me when there are no more jobs executing that logged on with a jobname of `COMPILE` — but only check every 10th time the task wakes up".

Viewing pending alarms (%SHOWALARM)

To see what alarms you currently have set, type `%SHOWALARM`. This will show you all of the alarms set by anyone logged on using the same session, user, and account name you are currently logged on to. For each alarm, you will see:

- an alarm number (you need this for the `%DELETEALARM` command, documented below);
- the time the alarm is set to "go off" or the expression which this alarm is set to test;
- the message this alarm will send when it "goes off".

If you say `%SHOWALARM;ALL`, you will also see:

- who set the alarm;
- who the message is to be sent to;
- any other special options set (`;CHECKEVERY=` or `;NOHP`).

If you have SM capability, you can use the `;FROM=` option to see alarms set by other users. For example:

```
%SHOWALARM;FROM=@.&ALL
```

will show you all alarms set by any users, and all parameters of those alarms.

Deleting pending alarms (%DELETEALARM)

To delete pending alarms, first do a [%SHOWALARM](#) (documented above) to find out the alarm number. Then simply type `%DELETEALARM` and the number of the alarm you wish to delete; for example, to delete alarm #5:

```
%DELETEALARM 5
```

Unless you have SM capability, you can only delete alarms that were set while logged on with the same session, user, and account name you are currently logged on with (these are the only alarms that `%SHOWALARM` shows you).

%ALIAS, %UNALIAS

Syntax:

```
%ALIAS [aliasname command]
```

```
%UNALIAS aliasname
```

Examples:

```
%ALIAS WT WARNF TOM,@.@;From Ron:
%WT This is a test of an alias...
%ALIAS SS SHOWJOB JOB=@S
%SS
%UNALIAS SS
%ALIAS
```

The `%ALIAS` commands allows you to create your own commands, sort of like UDCs or command files. So, why have a separate command when you could just use UDCs or command files?

- UDCs are hard to modify.
- Command files, while easier to change, require a separate file for each new command (on MPE/iX, the smallest amount of space a file can take is 16 sectors or 4096 bytes!)
- Both UDCs and command files require you to create a file, which means stopping what you're doing long enough to run some editor.

Aliases are created by simply typing "`%ALIAS`", an aliasname (which can be anything that an MPEX variable name can be), and the command to be executed when you enter that alias. When you log off, all of your aliases are automatically deleted. This makes it very convenient for you to create a simple, immediate shortcut for some complex sequence that you need to do repetitively (perhaps only for one day).

For example, suppose you're testing some program, and you have to run it with a long `;INFO=` string to test it, but not when it is put into production (presumably, you already have a UDC or command file that does this). You could say something like:

```
%ALIAS TESTIT RUN MYPROG;INFO="TESTDATA1 TESTDATA2"
```

Now, all you have to do is say

```
%TESTIT
```

MPEX COMMAND REFERENCE: %ALLOCATE

But, what if you need to run it with different parms? You could create several aliases, but a better solution would be

```
%ALIAS TESTIT RUN MYPROG;INFO="TESTDATA1 TESTDATA2";PARAM=
```

Then you can say

```
%TESTIT 20          (* to run with PARM=20 *)  
%TESTIT 32          (* to run with PARM=32 *)
```

As you can see, everything you type after the alias name is appended to the command that corresponds to that alias.

To see a list of your current aliases, simply type

```
%ALIAS
```

without any parameters.

To remove an alias, type

```
%UNALIAS aliasname
```

%ALLOCATE

Syntax:

```
%ALLOCATE PROCEDURE, { procname }  
  
%ALLOCATE [PROGRAM,] { fileset }  
           [;LIB= { G | P | S }]
```

Examples:

```
%ALLOCATE MY.OWN.PROGRAM;LIB=G  
%ALLOCATE PROCEDURE, VECMMND
```

MPEX has enhanced the normal MPE :ALLOCATE command to allow the use of the traditional ;LIB=G or ;LIB=P parameter (;LIB=S is also supported but has no effect as it is the default). MPE does not allow specifying ;LIB=G or ;LIB=P on the allocate command.

As with many other MPEX commands, MPEX allows you to specify a fileset instead of a single file name. Note that the file attribute "ISCMPPROG" is quite useful here in determining which files even NEED to be allocated.

For example, when installing our software, MPEX %HOOKs several programs and places a copy in the VESOFT account. To %ALLOCATE all of these programs using ;LIB=P, so that the HOOK procedure is loaded, type the following command:

```
%ALLOCATE @.PUB.VESOFT(ISCMPPROG);LIB=P
```

%ALTFILE

Syntax:

```
%ALTFILE fileset
```

General Characteristics:

```
[;CAP= [+|-] progcapability [,...]]
[;CODE= filecode]
[;KEEPAMDATES]
[;LOCKWORD= lockword]
[;MAXDATA= progmaxdata]
[;STACK= progstacksize]
```

File Ownership and Access:

```
[;CREATOR= username]
[;CREATOR= username [.accountname]] (POSIX)
[;NOCHECK] (POSIX)
[;DELACD]
[;GROUPID= groupid] (POSIX)
[;LOCAL[GROUPID]] (POSIX)
```

Physical Structure:

```
[;BINARY|ASCII]
[;BLKFACT= { blockingfactor|BEST }]
[;DEV= { devicenumber|deviceclass }]
[;EXTENTS= maxextents]
[;FIXED|VARIABLE|UNDEF|BYTE]
[;FLIMIT= flimitspecifier]
[;INITEXTENTS= initialextents]
[;KEEPTRAIL]
[;REC= newrecordlength]
[;SQUEEZE]
[;XLTRIM]
```

The %ALTFILE command can change many of a file's attributes. It can:

- Save disk space (XLTRIM, FLIMIT, SQUEEZE);
- Better manage the disk space you have by moving files from drive to drive (DEVICE);
- Make room in files (MPE or KSAM!) that are out of space (FLIMIT);
- Let you transfer ownership of files from one user to another (OWNER=), including ACCOUNT (GROUPID=) information on MPE/iX systems that support POSIX as well as removing ACDs (DELACD) that would override normal MPE security;
- Change the attributes of improperly :PREPped program files, preventing run-time capability errors and stack overflows (CAP, MAXDATA, and STACK);
- Alter the file codes and lockwords of files or filesets (CODE, LOCKWORD).
- Change file's internal structure (FIXED, VARIABLE, UNDEF, BYTE)
- Change the file from ASCII to BINARY (or the other way around, if you prefer)
- Change the length of each record, optionally keeping trailing SPACES for ASCII files, if desired. (REC=... and ;KEEPTRAIL)

All of these options are described in detail below. Note that you can combine any of the above options on one command, as well as use these parameters on a %COPY command (see the [%COPY command](#) for more details and examples). e.g.

```
%ALTFILE @.DATA; FLIMIT=EOF*1.5;XLTRIM;KEEPAMDATES
```

or

```
%COPY MYPROG, YOURPROG; CAP=+PH,DS; MAXDATA=30000
```

POSIX note: %ALTFILE security considerations

Ordinarily, an account manager can use the %ALTFILE command against any file within any group of his own account. On systems supporting POSIX, files have an attribute called the FILEGROUP (which is the ACCOUNT of the OWNER of the file). Since a system manager can %RENAME a file to another account, it is possible to have a file that an account manager cannot alter because the file belongs to a FILEGROUP that is different from the account manager's FILEGROUP.

Changing file blocking factors (;BLKFACT=)

Syntax:

```
%ALTFILE filespec;BLKFACT={expression|BEST}
```

Examples:

```
%ALTFILE @.DATA(ISASCII); BLKFACT=BEST
%ALTFILE MYFILE; BLKFACT=128
```

This keyword of the %ALTFILE command lets you change a file's blocking factor (or the blocking factors of a set of files). You can set the new blocking factor to a given number or let MPEX choose the best possible value (in terms of disk space) by specifying ;BLKFACT=BEST.

Note: %ALTFILE...;BLKFACT=... works on all systems, including MPE/iX; however, since blocks are handled quite a bit differently under MPE/iX, this discussion and the advantages described in it might not apply to MPE/iX machines. If you're using MPE/iX, you might want to skip the %ALTFILE...;BLKFACT discussion altogether.

Saving disk space

Bad blocking factors can waste disk space. For example, typing

```
:BUILD RPTLIST;REC=-133,,,ASCII;CTL
```

will make MPE assign a default blocking factor of 1, which will waste 47.5% of the disk space used by the file! MPE's default blocking factors, which are optimized to save memory (no longer a scarce resource) can often result in very bad disk space usage.

Typing a command such as

```
%ALTFILE @.DATA(ISASCII); BLKFACT=BEST
```

will set the blocking factors of ASCII ("plain vanilla") files in the DATA group to the BEST blocking factor — the best in terms of disk space usage. If several different blocking factors would result in the same file size, MPEX chooses the HIGHEST one to optimize access speed.

The blocking factor can be BEST, an integer constant or an EXPRESSION calculated from various file parameters (the same ones available in selection criteria). For instance, you might want to set a file's blocking factor to 8192 divided by the record size (in bytes):

```
%ALTFILE @.DATA(ISASCII); BLKFACT=8192/RECSIZE
```

(This, of course, will set the blocking factor to the highest value such that the resultant block size is ≤ 8192 bytes.) If we also want to limit the blocking factor to at most 100, we could say

```
%ALTFILE @.DATA(ISASCII); BLKFACT=MIN(8192/RECSIZE,100)
```

This ability to specify expressions as %ALTFILE parameters is available on the BLKFACT=... and FLIMIT=... keywords; it isn't (currently) available on any of the others, primarily because we haven't found any case in which we think it'll be useful. If you can think of one, let us know.

Speeding up file access

In normal buffered file access (the default access mode), the file system does disk I/Os in units of one block — the blocking factor is the number of records in each block. If a 30,000-record file which has blocking factor 3 is read sequentially, the file system will have to issue 10,000 disk I/Os. If the file has a blocking factor of 100, it could be read in only 300 disk I/Os; although each 100-record I/O would be somewhat slower than each 3-record I/O, you could still realize a 10-fold performance improvement.

Therefore, it can be quite desirable to increase the blocking factors of some of your MPE or KSAM files — especially ones that are accessed sequentially.

For instance, saying

```
%ALTFILE DATAFILE; BLKFACT=100
```

will increase DATAFILE's blocking factor to 100; saying

```
%ALTFILE @UDC@; BLKFACT=EOF
```

will set the blocking factor of all the files that have "UDC" in their file names to be equal to their current EOF, thus making it possible (on MPE/V) to read the entire file in one I/O (this only works if the EOF is 255 or less).

It's often a good idea to increase the blocking factor of your UDC files because they must be read in their entirety every time a user logs on — increasing their blocking factors can thus greatly speed up logon time. (Of course, the UDC files must be un-:SETCATALOGed before reblocking and re-:SETCATALOGed afterwards.)

Note that you pay for this speed by higher memory usage (as a rule, the system keeps two blocks of memory allocated for every accessor of a disk file – if your file has blocking factor 60 and record size 40 words, each of the users of the file will have 4800 extra words of memory

allocated for him). If you find this a problem (and most people don't), you can easily lower blocking factors.

WARNING

Like most %ALTFILE options, %ALTFILE...;BLKFACT=BEST doesn't change the data in your file — it only changes the format in which the data is kept on disk. Most of your programs shouldn't care what a file's blocking factor is; however, a few programs (especially system utilities) do care, and sometimes make assumptions about a file's blocking factor.

A good example of this is the MPE :SEGMENTER subsystem — it assumes that all USL, RL, and SL files have blocking factor 1. If you change a USL file's blocking factor, :SEGMENTER won't be able to access the file, not because the blocking factor change somehow "ruined" the data in the file, but because :SEGMENTER's assumptions are no longer valid.

Because of this, MPEX doesn't let you change the blocking factor of files whose file codes are 1024 or greater — these file codes are generally reserved for MPE or HP subsystems, which usually make assumptions about the blocking factors of these files. This rule prevents you from accidentally rendering some of your USL files, program files, etc. unusable.

It is still possible, though, that one of your own programs (or a program supplied by a third-party vendor) can't handle files whose blocking factors have been changed. Naturally, MPEX doesn't know this, and will reblock such a file if you tell it to (remember, "a computer does what you tell it to do, not what you want it to do").

If this happens to you — if you change a file's blocking factor (or its file limit) and your program doesn't like this — don't panic! Just use %ALTFILE to change the blocking factor back to whatever it was before.

Similarly, before doing a fileset %ALTFILE...;BLKFACT=... check to see if there are any special files in the fileset — if there are, try altering one of them first and seeing if the program that normally accesses it can still do so.

If you find that the program complains, you can easily exclude that file (and others like it) by saying something like

```
%ALTFILE @.@(CODE<>"557" and CODE<>"558"); BLKFACT=BEST
```

Note: %ALTFILE...;BLKFACT=BEST also forbids changes of files with file code 111 [QEDIT files] but allows changes of files with file code 1052 [EDTCT files].

Changing program file capabilities (;CAP=)

Syntax:

```
%ALTFILE fileset;CAP={ [+ | - ] [IA,BA,PH,PM,MR,DS]}
```

Examples:

```
%ALTFILE AP010P; CAP=+PH,DS
%ALTFILE @.UTIL.SYS(ISPROG); CAP=-PM,PH,MR,+DS
```

In order to do certain things — create son processes, acquire extra data segments, lock multiple files/databases or use privileged mode — a program must be :PREPped (or, for MPE/iX native mode programs, :LINKed) with the appropriate capabilities.

What if you omit the right capability? With MPEX you don't need to re-:PREP the program (or recompile it if you've lost the USL); you can just give it the capabilities that it should have.

For instance, saying:

```
%ALTFILE AP010P; CAP=+PH,DS
```

will add to the program file AP010P the capabilities PH and DS (without removing any of the capabilities it already possesses).

Similarly, you can use %ALTFILE to remove capabilities from a program. For example:

```
%ALTFILE @.UTIL.SYS(PROG.PMCAP); CAP=-PM
```

will remove PM capability from all the UTIL.SYS programs that have PM.

Specifying all capabilities without a + or - (e.g. %ALTFILE XFILE; CAP=PH,DS) is the same as specifying them with a +; it adds the capabilities while retaining any other capabilities the program might have. However, specifying ;CAP=PH,-MR,DS is equivalent to ;CAP=+PH,-MR,-DS. Similarly, specifying ;CAP=-BA,PM,PH,+MR,DS would be treated as ;CAP=-BA,-PM,-PH,+MR,+DS. In other words, unsigned capabilities retain the same sign as the previous capability listed.

As we implied above, %ALTFILE...;CAP= also works on MPE/iX native program files.

Related MPEX features

In addition to being able to alter a program file's capabilities, MPEX lets you do a couple of other things that MPE doesn't let you do easily (or at all):

- List a program file's capabilities:

```
%LISTF MYPROG.DEV,5
FILENAME  MAXDATA  STACK  DLSIZE  DBSIZE  #SEGS  CAPABILITIES
MYPROG      30000   2048      0    4604    28    BA IA PM DS PH
```

A number of important file parameters, including the MAXDATA, capabilities, the program's global storage size (in this case 4604), and the total segments (28) are shown.

Another way to see this (and much more) information about program files is with the %PROGINFO command (described later in [this manual](#)).

- Find all programs with (or without) a specific capability:

```
%LISTF @.@.@ -@.@.VESOFT - @.PUB.SYS(PROG.PMCAP),5
%LISTF / - /VESOFT/ - @.PUB.SYS(PROG.PMCAP),5
```

will find for you all the programs in the system (except for those in the VESOFT account or in PUB.SYS) that are allowed to use PM capability.

These features work hand in hand with the %ALTFILE...;CAP= command.

Changing file codes (;CODE=)

Syntax:

```
%ALTFILE filespec;CODE=filecode
```

Examples:

```
%ALTFILE MYDB; CODE=400
%ALTFILE MYDB01; CODE=401
```

In addition to all the other things, MPEX also lets you change a file's file code.

Why would you want to do this? There are several possible reasons:

- Some (though by no means all) communication programs (especially PC-to-HP3000 transfer utilities) always build HP3000 files with code 0 — if you download a file to a PC and then upload it back, you'll find that you've lost the filecode. %ALTFILE...;CODE= can put it back.
- Since the file system normally doesn't care much about a file's filecode — a normal program can read a file with one filecode just as well as with another — the filecode can be a convenient way of "marking" files. Thus, you can set some of your data files to a particular filecode so they look more distinct on :LISTFs and so they can be selected with MPEX's selection conditions.

An interesting idea is for marking "ephemeral" files — files that you know you'll only be using for a few days and want to automatically purge, say, at the end of the week. Whenever you build such a file you might %:ALTFILE...;CODE= it — then, even if you forget about it, you can easily find it when you want to clean your disks.

If you want to get really fancy, you might have a convention to set the filecode to 1 if you want to keep the file for one day, to 2 if you want to keep it for two days, etc. Then, you can periodically say

```
%PURGE @.@(BETWEEN(INTCODE,1,14) and ACCDATE<TODAY-INTCODE)
```

This will purge all the files whose code is between 1 and 14 and which haven't been accessed in that many days!

- You may also want to change a file's filecode if the code is currently negative (i.e. PRIV, quite likely an IMAGE file). If you want to manipulate IMAGE files using a program that normally can't handle them (e.g. DSCOPY or DISKED5), you might want to alter the file's filecode to a positive value, do whatever you need to do, and then alter the filecode back.

Needless to say, you can get into a lot of trouble this way. Naturally, MPEX only allows a database's creator, its account manager or the system manager (all three of whom can do anything to the database anyway) to do this operation; we also encourage people to be very careful with what they do with the internal structures of their precious databases.

However, under some circumstances (with somebody who knows what he's doing), this can be a rather useful feature.

Note, however, that changing a filecode to some value doesn't magically make that type of file. For instance, if you change a source file's filecode to PROG, the file won't become a valid program file; the :RUN command will just give you some nasty error message when you try running it. If you change the code to some other system-defined value (e.g. USL, SL, etc.), whatever program accesses the file will probably complain. Use this option carefully.

Changing file ownership (;CREATOR= and ;GROUPID=)

Syntax:

```
%ALTFILE filesset;CREATOR=username
%ALTFILE filesset;CREATOR=username;NOCHECK
%ALTFILE filesset;CREATOR=username.acctname
%ALTFILE filesset;GROUPID=groupid
%ALTFILE filesset;GROUPID=groupid;NOCHECK
%ALTFILE filesset;LOCALGROUPID
%ALTFILE filesset;creator=username;DELACD
%ALTFILE filesset;GROUPID=groupid;DELACD
```

Examples:

```
%ALTFILE @.@(CREATOR="TOM"); CREATOR=MANAGER.SYS
%ALTFILE @.@(CREATOR<>"DBA" and ISPRIV); CREATOR=DBA
%ALTFILE @.@.SYS;GROUPID=SYS
%ALTFILE @.@@;LOCALGROUPID
%ALTFILE @.SOURCE;CREATOR=NOBODY.ATALL;NOCHECK
```

A file's creator ID is often very important:

- MPE's :RENAME, :RELEASE, :SECURE, and :ALTSEC commands can only be performed by a file's creator (although MPEX's %RENAME, %RELEASE, %SECURE, and %ALTSEC don't have this restriction).
- As a rule, many database utilities can only be run by a file's creator (unless you use maintenance words).
- An ordinary :RESTORE (without ;CREATE) requires the file's creator ID to exist in the system.

Say that one of your programmers left your company; naturally, the first thing you did is :PURGEUSERd his user ID. Now, all his files are "orphans" – they can't be :RENAMEd, :RELEASEd, :SECURED or :ALTSECed; his databases can't be DBUTILd; and unless you're careful enough to put a ;CREATE on all your :RESTOREs (e.g. after your next RELOAD), they won't be restored.

What you really need to be able to do is to "re-assign" all of these "orphans" to another user. For instance, you might say

```
%ALTFILE @.@(CREATOR="TOM"); CREATOR=MANAGER
%ALTFILE @.@(CREATOR="TOM"); CREATOR=MANAGER.SYS;GROUPID=SYS
```

This will find all the files in your account that have creator ID TOM and change their creator ID to be MANAGER (or MANAGER.SYS). Similarly,

```
%ALTFILE @.@(CREATOR<>"DBA" and ISPRIV); CREATOR=DBA
```

will change all your databases (which don't already have a creator ID of DBA) to be "owned by" DBA.

Related POSIX features

Just changing the name of the creator, however, doesn't completely change the "ownership" of a file on a POSIX system. On these systems, a file is "owned" by the CREATOR and the account manager of the GROUP in which the file resides. Note that this is NOT the MPE group, but

rather the POSIX "GROUPID", (which has its roots in the UNIX operating system) and is, in reality, an MPE ACCOUNT name.

When a file is RENAMED from one account to another (using either MPEX or MPE itself), the CREATOR and GROUPID don't change, so the CREATOR and account manager of the account where the file was originally built retain "ownership" of the file. In order to enforce this, MPE may place an Access Control Definition (ACD) on a file that has been renamed.

In order to change the ownership of all files on your system to the account where each file now resides, use the ;LOCALGROUPID (which can be abbreviated to ;LOCAL) and ;CREATOR keywords as shown here:

```
%ALTFILE @. @. @. @. ;LOCALGROUPID;CREATOR=MANAGER;DELACD
```

(This presumes that all of your account managers have a user ID of MANAGER; however, many sites use MGR as the account manager.) If you want to set the ownership of a set of files to a particular GROUPID (for instance, to secure the source of all of your job streams in a particular account), you can use the following:

```
%ALTFILE @. JOBS. PROD;GROUPID=SYS;CREATOR=MANAGER. SYS
```

This would effectively make the system manager the "owner" of all of your job streams in the PROD account.

Note: ;GROUPID= and ;LOCALGROUPID are mutually exclusive — you can specify one or the other, but not both in the same command.

As the examples above show, the ;CREATOR= keyword allows you to specify the ACCOUNT as well. If no account is specified, MPEX will assign the account name where the file resides as the account name of the creator. In addition to specifying the MPE user ID and account, you can use an asterisk to indicate yourself or an equals sign to indicate that the field should retain whatever value it currently contains.

By "yourself", we mean the MPE user and account information for your current logon. For instance, if you are logged on as MANAGER. ACCTING and type:

```
%ALTFILE @. DATA;CREATOR=*.*
```

then it would be equivalent to entering ;CREATOR=MANAGER. ACCTING.

The "=" character retains whatever value is currently in the field, so if a file was "created" by CLERK. PAYROLL, then this command:

```
%ALTFILE @. DATA;CREATOR=MGR. =
```

would be the same as saying ;CREATOR=MGR. PAYROLL.

Note: To maintain compatibility with MPE, MPEX generally will not let you change a file to be "owned" by a non-existent user. This restriction can be overridden in MPEX by supplying the ;NOCHECK keyword on the %ALTFILE command.

Related MPEX features

We've already seen how the `CREATOR=` selection condition can work very well with the `%ALTFILE...;CREATOR=` command. Another relevant feature is:

```
%LISTF MYFILE.PUB.SYS,3
FILENAME  CREATOR  CRE-DATE  MOD-DATE  MOD-TIME  ACC-DATE  RST-DATE
MYFILE    MANAGER  11 AUG 91  24 DEC 91  10:52 AM  05 FEB 91  30 DEC 91
```

As you see, `%LISTF... ,3` shows you creator IDs, times, and dates.

Another interesting use of MPEX selection conditions in `%ALTFILE...;CREATOR=` is the following:

```
%ALTFILE @.@(NOT USEREXISTS(CREATOR+"."+ACCOUNT)); CREATOR=MGR
```

Wow! What does all this mean? Well, the selection condition here is:

```
NOT USEREXISTS(CREATOR+"."+ACCOUNT)
```

We're executing the MPEX built-in function `USEREXISTS(...)`, and passing to it the file's creator, concatenated with a "." and the file's account. The `USEREXISTS` function (which you can look up in Appendix to this manual) returns TRUE if a particular MPE user.account exists. We said "`NOT USEREXISTS...`" because we want to select those files whose creator does NOT exist. Therefore, MPEX will change to "MGR" the creator of all files (in your logon account) whose current creator doesn't exist in the file's account.

Similarly, if you have SM capability, you could say

```
%LISTF @.@.(NOT USEREXISTS(CREATOR+"."+ACCOUNT)),3
```

and find all the files in the system whose creators don't exist.

Moving files from disk to disk (;DEV=)

Syntax:

```
%ALTFILE fileset;DEV=devicenumber|deviceclass
```

Examples:

```
%ALTFILE CUST?#.DB(DBSETTYPE="D"); DEV=2
%ALTFILE CUST?#.DB(DBSETTYPE<>"D"); DEV=3
%ALTFILE @.@(ONDEVICE(1)); DEV=APPDISC
```

Which disk drives a file resides on can have a substantial impact on system performance. One popular theory holds that having frequently-accessed files on the same disk drive can increase contention for that drive and therefore slow down access to it. Therefore, many people like to:

- Keep frequently-accessed IMAGE master and detail files on different drives;
- Keep KSAM data files on different drives than their key files;
- Minimize the amount of non-system files that reside on device 1 (since it is already heavily used for system access, directory access, and sometimes for virtual memory access).

MPEX is the most convenient tool for doing this sort of thing.
For example, saying

```
%ALTFILE CUST?#.DB(DBSETTYPE="D"); DEV=2
%ALTFILE CUST?#.DB(DBSETTYPE<>"D"); DEV=3
```

will move all the detail datasets of the CUST database (the selection condition DBSETTYPE="D" will select only the detail datasets) to disk drive #2, and all the non-detail (manual or automatic) datasets to disk drive #3. Similarly,

```
%ALTFILE @.@(ONDEVICE(1)); DEV=APPDISC
```

will move all files in your account that have at least one extent on device #1 to device class APPDISC (which might be configured to include all disk drives except for device 1).

What you should know about extents

Before moving a lot of files, you must be aware of some little-known facts about the way disk files are built.
By default, a disk file is built on device class DISC, which is (usually) configured to include all the disk drives in the system. This means that each of a file's extents could conceivably be on a different disk drive.

For instance, if you use MPEX's %LISTF...,4 to look at a file's extent map, you might see the following:

-----FILE-----	EXTENTS	-BLKFACT-	-----SECTORS-----	DEVICE
LABEL				
NAME	CODE	NUM MAX NOW BEST USED NOW SAVABLE	CLASS	LDEV
LOG0679		16 16 1 8192	DISC	2
Dev/Sector:		2/%00000241110 3/%00000105237	1/%00000544165	
Dev/Sector:		1/%00000537652 3/%00000523744	4/%00000534046	
Dev/Sector:		3/%00000531514 2/%00000544520	4/%00000526374	
Dev/Sector:		2/%00000540024 1/%00000540652	1/%00000535347	
Dev/Sector:		1/%00001120104 2/%00001310054	4/%00000763606	
Dev/Sector:		1/%00001137722		

As you see, this file has 6 extents on device 1, 4 extents on device 2, 3 extents on device 3, and 3 extents on device 4.

However, your :STORE listing will show this file as being "on device #2" — :STORE always shows the device number of the file label! MPE/iX has a :LISTF...,3 (not the same as MPEX's %LISTF...,3!) that shows the "DISC DEV #", but this is actually the disk that the file label is on; the beginning of the file itself may be on a completely different disk!
Therefore, when you're figuring out which files should be moved, be sure that you have all the data! Do a %LISTF...,4 first to see exactly on which disk drive a file is located — even if you see a data and key file which :STORE shows as being on different disk drives, it may be that they're actually mostly on the same device.

Another relevant feature is the `ONDEVICE(devnumber)` selection criterion we mentioned above. Saying

```
%ALTFILE @.DATA(ONDEVICE(1)); DEV=APPDISC
```

will select (and move) those files that have at least one extent on device 1. Thus, any files which are even partially on the system disk will be moved.

The criteria that are related to file location are:

<code>ONDEVICE(<i>devnumber</i>)</code>	TRUE if the file has at least one extent on the given device.
<code>DEVICESECTORS(<i>devnumber</i>)</code>	An integer that indicates how many sectors of the file are actually on device <i>devnumber</i> .
<code>DEVICE=...</code>	A string indicating the DEVICE CLASS (e.g. "DISC", "SYSDISC", "MEBER2" or "2") — a device class may also be a device number.
<code>FLABLDEV=...</code>	An integer indicating where (on which device) the file label is.

For instance, some interesting commands might be:

```
%ALTFILE @.DATA(DEVICESECTORS(1)>=SECTORS/10); DEV=2
```

(affects all files that have at least one tenth of their sectors are on device 1; moves them onto device 2)

```
%ALTFILE @.DATA(DEVICE="SYSDISC"); DEV=APPDISC
```

(moves to device class APPDISC all the files that were built on device class SYSDISC — not those files that happen to be on a device in that class, but those that were explicitly built to be on that device class)

What you should know about non-system volumes

When moving files from one device to another, you should also be aware of the effect non-system volumes have on files. Note that on an XL system, the SYSTEM volume set is called "MPEXL_SYSTEM_VOLUME_SET", while NON-SYSTEM volume sets are given an arbitrary name when they are created. On MPE/V systems, the SYSTEM volume set did not have any special definition while NON-SYSTEM volume sets were usually referred to as PRIVATE VOLUMES.

The important thing to understand about volume sets is that the VOLUME SET for a particular file is not determined by the file itself, but rather the GROUP in which the file resides determines the VOLUME SET for the file. While all of the files in a particular group reside on the same volume set, different groups within the same account may reside on different volume sets. For this reason, when you %RENAME a file from one group to another, you may get an error indicating that the file cannot be renamed across volume sets. In this case, the file must be %COPYd instead.

For example, let's take a system with eight drives: LDEVs 1,2,3, and 4 are the SYSTEM volume set; LDEVs 5,6,7, and 8 are a NON-SYSTEM volume set. On this system, the account DEMO has three groups, PUB, SYSTEM, and PRIVATE. PUB and SYSTEM are in the SYSTEM volume set and PRIVATE is on the NON-SYSTEM volume set.

To move the file `SHOW.PRIVATE.DEMO` to `SHOW.PUB.DEMO`, you must use the %COPY command instead of the %RENAME command since these groups are in different volume sets. For the same reason, you cannot `%ALTFILE SHOW.PRIVATE.DEMO;DEV=2` but you could `%ALTFILE SHOW.PRIVATE.DEMO;DEV=8`. In general, however, you could `%ALTFILE SHOW.PRIVATE.DEMO;DEV=DISC` since the device class of DISC may be applied to devices in different volume sets (and is usually applied to all physical disk devices when configured). When you use `;DEV=disk` for a file on a non-system volume set, the file is placed on any disk within that set, so the file `SHOW.PRIVATE.DEMO` would exist on LDEV 5, 6, 7 or 8 after the above command was executed.

In order to move a group from a SYSTEM volume set to a NON-SYSTEM volume set, you need to use the `:NEWGROUP` and `:ALTGROUP` commands to define the group on the non-system volume set. The group must be empty when you do this, so you must `:STORE`, and `%PURGE` the files prior to moving the group, then `:RESTORE` the files after the group has been moved.

This works fine for a single group, but could become tedious for a large number of groups. Our VEAUDIT product can perform %ALTGROUP commands against groupsets, so moving groups from a system volume set to a non-system volume set is relatively easy (see the VEAUDIT manual for details).

Changing file maximum # of extents (;EXTENTS=)

Syntax: `%ALTFILE fileset;EXTENTS=maxextents`

Examples: `%ALTFILE MYFILE;EXTENTS=32`
`%ALTFILE @.DATA(EOF<FLIMIT and MAXEXTENTS<>32);EXTENTS=32`

Note: `%ALTFILE...;EXTENTS=...` works on all systems, including MPE/iX; however, since extents are handled quite a bit differently under MPE/iX, this discussion and the advantages described in it might not apply to MPE/iX machines. If you're using MPE/iX, you might want to skip the `%ALTFILE...;EXTENTS=...` discussion altogether.

A disk file may be built as any number (from 1 to 32) of chunks of contiguous disk space. Each such contiguous chunk is called an extent.

The most important thing about extents is that a file is always allocated extent by extent:

- If you build an entirely empty file — one that contains no data — an entire extent of the file will have to be allocated (if the file has file limit 100,000 and a maximum of 8 extents, then enough room for 12,500 records will be allocated even if none of them are actually used);
- Whenever a file is not full — its EOF is less than its FLIMIT — ½ of an extent is, on the average, wasted.

For instance, in that file with FLIMIT 100,000 and a maximum of 8 extents, on the average 6,250 records worth of data will be wasted — allocated but not used. For example, in the file

FILENAME	CODE	-----LOGICAL RECORD-----				----SPACE----		
		SIZE	TYP	EOF	LIMIT	R/B	SECTORS	#X MX
MYFILE		134B	FAC	13215	100000	21	13112	2 8

6,171 sectors of data (almost half!) are wasted — there are only 13,215 records in it, but since each extent has 12,500 records (100,000 / 8), MPE allocates enough disk space for 25,000!

What this really means is that:

- Whenever you expect a file to have EOF < FLIMIT, you should build it with a maximum of 32 extents, the largest value allowed by MPE. (MPE's default is 8, which is less efficient than 32.)

With 32 extents, the average amount of wasted space (½ of an extent) and the maximum amount of wasted space (1 full extent) will both be minimized.

For instance, if in the above example we say

```
%ALTFILE MYFILE;EXTENTS=32
```

then our file would look like:

FILENAME	CODE	-----LOGICAL		RECORD-----		-----SPACE-----			
		SIZE	TYP	EOF	LIMIT	R/B	SECTORS	#X	MX
MYFILE		134B	FAC	13215	100000	21	8195	5	32

Simply by increasing the maximum number of extents for the file (which doesn't decrease the room available in the file at all), we've saved 4,917 sectors or 37.5% of the file's disk space! There's still some "wasted" space, but it's a lot less now than it was before.

A good example of the use of MPEX filesets on the %ALTFILE command might be:

```
%ALTFILE @.DATA(EOF<FLIMIT and MAXEXTENTS<>32);EXTENTS=32
```

This finds all the files that are not full (EOF<FLIMIT) and don't already have a maximum of 32 extents and changes them to have a maximum of 32 extents.

Note that the MAXEXTENTS keyword (which indicates the maximum number of extents this file could have) shouldn't be confused with NUMEXTENTS (which indicates the number of extents the file currently has). MAXEXTENTS corresponds to the "MX" column on a :LISTF..., 2; NUMEXTENTS corresponds to the "#X" column.

Also note that sometimes when you ask for a certain maximum number of extents, MPE will actually build the file with a smaller maximum number of extents — for instance, %ALTFILEing a file with EXTENTS=32 might actually make the file have only 29 extents.

This is a feature of MPE which MPEX can do nothing about (the MPE :BUILD command often does the same thing); in any case, it shouldn't cause any problems, since the maximum number of extents only affects the disk space allocated to a file, not the behavior of programs accessing it.

Changing file flimits (;FLIMIT=)

Syntax:

```
%ALTFILE fileset; FLIMIT=flimitspecifier
```

Examples:

```
%ALTFILE @.DATA(INTCODE<>717); FLIMIT=EOF*1.5
%ALTFILE MYFILE.DATA; FLIMIT=FLIMIT-10000
```

Saving disk space

Whenever a file's FLIMIT is greater than its EOF, chances are that some disk space will be wasted. This wasted space can be minimized by increasing the number of extents to 32 (see [";EXTENTS="](#) in the %ALTFILE command); but even then, most of the time there will be some disk space that is allocated but not used.

Consider, for a moment, the following file:

FILENAME	CODE	-----LOGICAL RECORD-----			----SPACE----			
		SIZE	TYP	EOF	LIMIT R/B	SECTORS	#X	MX
MYFILE		134B	FAC	13215	100000 21	8195	5	32

As you see, it occupies 8,195 sectors of disk space; however, only about 6,900 of these (13215*134/256) really contain data — the rest are unused.

It's quite possible that MYFILE will never have 100,000 records; it might be quite safe to leave just enough additional room for, say, half as many records as are already in the file. All we need to do is say

```
%ALTFILE MYFILE; FLIMIT=EOF*1.5
```

and then the file will look like:

FILENAME	CODE	-----LOGICAL RECORD-----			----SPACE----			
		SIZE	TYP	EOF	LIMIT R/B	SECTORS	#X	MX
MYFILE		134B	FAC	13215	19822 21	7260	22	32

As you see, we've saved 935 sectors or 11% of the file's disk space. Only 319 sectors are now wasted (allocated but not used). And, chances are that the room left in the file (50% of its current size) should be plenty for any further expansion.

What if a file overflows?

To save disk space, we can lower file limits. However, what if your data file (MPE or KSAM) gets full? You built it with room for 20,000 records, so when you try to write the 20,001st, MPE doesn't let you. What can you do?

MPEX's %ALTFILE...;FLIMIT=... can easily be used to increase file limits as well as decrease them.

```
%ALTFILE MYFILE; FLIMIT=EOF*1.5
```

will decrease MYFILE's FLIMIT if it's currently greater than EOF*1.5; it will increase the FLIMIT if it's currently less than EOF*1.5. For instance, if MYFILE currently looks like

FILENAME	CODE	-----SIZE	-----TYP	LOGICAL EOF	RECORD-----LIMIT	R/B	-----SPACE-----SECTORS	#X	MX
MYFILE		144B	FA	19974	20000	16	11259	32	32

we can see that it's almost full. If we say

```
%ALTFILE MYFILE; FLIMIT=EOF*1.5
```

it will now look like:

FILENAME	CODE	-----SIZE	-----TYP	LOGICAL EOF	RECORD-----LIMIT	R/B	-----SPACE-----SECTORS	#X	MX
MYFILE		144B	FA	19974	29961	16	11682	22	32

Of course, we could use any of the other ;FLIMIT=... options as well; we could say one of:

```
%ALTFILE MYFILE; FLIMIT=30000
%ALTFILE MYFILE; FLIMIT=FLIMIT+10000
%ALTFILE MYFILE; FLIMIT=FLIMIT*1.5
```

any of which will set the file limit to 30,000. Usually, the FLIMIT=EOF*xxx construct is the best since it very reasonably represents what you'd probably like the new file limit to be.

In fact, you might want to have a nightly job stream that says:

```
:JOB MAKEROOM,...
:RUN MAIN.PUB.VESOF
!ALTFILE @.DATA(EOF>0.8*FLIMIT); FLIMIT=EOF*1.5
EXIT
:EOJ
```

This will find all the files that are more than 80% full ($EOF > 0.8 * FLIMIT$) and set their file limits to 50% more than their current EOFs.

Alternatively, you might want to expand your data file before running a program that adds a lot of data to it, and then contract it after the program's done:

```
:JOB MYRPT,...
:RUN MAIN.PUB.VESOF;PARM=1
ALTFILE X77U.DATA; FLIMIT=EOF*5
RUN X77P.PUB;LIB=P
ALTFILE X77U.DATA; FLIMIT=EOF*1.1
EXIT
:EOJ
```

This will set the FLIMIT to 5 times the EOF (EOF*5) before the program is run; after the program's done, it'll set the FLIMIT to 10% more than the new EOF. It'll take some extra time (not too much), but will both optimize your disk space usage AND make sure that the file doesn't overflow.

WARNING

The %ALTFILE...;FLIMIT=... option tells the file system what the maximum number of records in the file is supposed to be. Usually, this does exactly what you want it to do — saves disk space or makes more room for future expansion.

However, some programs make assumptions about how large a file is supposed to be. Say that you have a program that 'knows' that the FLIMIT recorded somewhere within the file itself (the MPE :SEGMENTER subsystem is like that).

Then, when you ask MPEX to change the file's FLIMIT, MPEX will be very happy to do exactly what you asked — change the FLIMIT. However, the data inside the file (known only to the program that manages the file) will naturally not be changed.

This is why MPEX doesn't let you do %ALTFILE...;FLIMIT=... of RL, SL, or USL files — if you try to increase the file limit, it'll have no effect; if you try to decrease the file limit, you'll confuse :SEGMENTER and when you next try to add something to the file, the file might even get corrupted.

Even if MPEX let you %ALTFILE...;FLIMIT=... of RLs, SLs, and USLs, this wouldn't be a "bug" — MPEX would be doing exactly what you told it to do.

However, since you would never really want to change an RL's, SL's, or USL's file FLIMIT (because of the problems with :SEGMENTER), MPEX specially checks for this case.

You might have other files just like this that MPEX can't possibly know about — for instance, files manipulated by your own programs or some third-party vendor products. Whenever a file contains — somewhere in its data — information on its own flimit, MPEX's %ALTFILE...;FLIMIT=... could cause problems, since it'll change the FLIMIT but not the data-resident information.

MPEX will gladly do what you ask, but the program that later accesses the file won't like it (although the file's data will be exactly the same as before).

To avoid problems with this, you should do the following:

- Don't panic — remember, you could always do another %ALTFILE...;FLIMIT=... to change the FLIMIT back to what it was before.
- If you suspect that one of your programs may not like %ALTFILE...;FLIMIT=... of its data files, try this on one file (rather than doing an %ALTFILE of an entire group or account).
- If you do find that some files shouldn't have %ALTFILE...;FLIMIT=... done to them, you can easily exclude them from your filesets; for instance, if files with code 557 and 558 shouldn't have their FLIMITs changed, you can simply say

```
%ALTFILE @.DATA(INTCODE<>557 and INTCODE<>558); FLIMIT=EOF*1.5
```

Changing file initially allocated extents (;INITEXTENTS)

Syntax:

```
%ALTFIL fileset;INITEXTENTS=initialextents
```

Examples:

```
%ALTFIL MYFILE;INITEXTENTS=32
%ALTFIL @.DATA(NUMEXTENTS<>MAXEXTENTS) ;INITEXTENTS=MAXEXTENTS
```

%ALTFIL...;INITEXTENTS= lets you specify the initial number of extents to be allocated to the new file. This can be useful in several ways:

- You may want to pre-allocate space for the whole file so that if there's not enough room for it on disc, you'll know now rather than when your program tries to allocate a new extent for the file (which might be in the middle of your batch processing). To do this, you might do an %ALTFIL...;INITEXTENTS=32, which will allocate the file with the maximum number of extents (even if that is less than 32).
- On MPE/iX, you may want to allocate the file as one contiguous chunk of disk space — this can be done with an %ALTFIL...;EXTENTS=1;INITEXTENTS=1.
- You might want to do an %ALTFIL preserving whatever extent-allocation parameters were specified for the current files — you can do this by specifying ;INITEXTENTS=NUMEXTENTS, which means that the new file is to be allocated with an initial number of extents equal to the number of extents that are currently allocated.

Preventing changes to file access/modify dates (;KEEPAMDATES)

Syntax:

```
%ALTFIL fileset;...;KEEPAMDATES
```

Examples:

```
%ALTFIL @.DATA;XLTRIM;DEV=2;KEEPAMDATES
```

When you do an %ALTFIL, the file's last access and last modify dates will be set to today's date. This is rather typical MPE behavior — after all, you're both accessing and changing the file, so the last access and the last modify dates should be updated.

However, sometimes you do not want these dates to be updated. In particular, you might not want to update the last access date because you use it for archival purposes; you might periodically say something like

```
%MPEXSTORE @.@.@-@.@.SYS-@.@.TELESUP (ACCDATE<TODAY-180) ;*TAPE;PURGE
```

to clean up your system and save disk space. If you do this, you'd like to update the last access date as rarely as possible — update it only when somebody actually tries to access the file for something productive.

If you merely say

```
%ALTFIL @.DATA;XLTRIM
```

you will have all the DATA group files marked as having been accessed today.

Similarly, if your partial backups take a long time, you might not want to update the last modify date when you do something as relatively non-intrusive as, say, an %ALTFIL

@.DATA;DEV=... — the files' contents haven't actually changed, so there's no good reason to force all the altered files to be backed up.

If you say

```
%ALTFILE @.DATA;DEV=2;XLTRIM;KEEPAMDATES
```

then MPEX will perform all the alterations without updating the last access and last modify dates. If you want this to be the standard behavior of the %ALTFILE command, you can say

```
:SETJCW MPEXALTFILEKEEPAMDATES=1
```

— this tells MPEX that the %ALTFILE command should never update the last access and last modify dates of files even if ;KEEPAMDATES is not specified.

If you're going to use the SETJCW command, you should probably put it into the MPEXMGR.PUB.VESOF file so that it comes into effect for all users in the system.

Note that for technical reasons, MPEX will always update the last access and modify dates of a KSAM key file when a KSAM file is %ALTFILEd.

Changing file lockwords (;LOCKWORD=)

Syntax:

```
%ALTFILE fileset;LOCKWORD=[ lockword]
```

Examples:

```
%ALTFILE @.UTIL.SYS(PROG.PMCAP); LOCKWORD=SECRET
%ALTFILE @.PRIV.TELESUP; LOCKWORD=!!RANDOMNAME
```

The %ALTFILE...;LOCKWORD= option lets you (you guessed it) alter a file's lockword (or the lockwords of all the files in a fileset).

POSIX

note: Since POSIX files cannot have lockwords, this keyword will not work with a POSIX fileset.

The advantages of %ALTFILE...;LOCKWORD= over a :RENAME (the normal way of altering a file's lockword) are:

- you can %ALTFILE...;LOCKWORD= entire filesets;
- you can %ALTFILE...;LOCKWORD=!!RANDOMNAME to assign a file (or all files in a fileset) a RANDOM lockword. (This is just a special case of the RANDOMNAME function [described in Appendix] which can be used for setting random lockwords, random MPE passwords, etc.)

For example, you can say

```
%ALTFILE @.@.CONTRIB(PROG.PMCAP); LOCKWORD=XYZZY
```

to set a lockword on all the program files in the CONTRIB account that have PM capability.

To assign a random lockword to each of the files in PRIV.TELESUP, use the following example:

```
%ALTFILE @.PRIV.TELESUP; LOCKWORD=!!RANDOMNAME
```

By assigning a random lockword to each utility in `PRIV.TELESUP`, you reduce the possibility that someone could run one of these programs without authorization.

Even if one of the lockwords is discovered, the rest of the utilities are protected since they each have a different lockword. When you need to use one of these utilities, log on with System Manager (SM) capability and either run the program from MPEX (where we will automatically supply the lockword) or use MPE's `LISTF` modes -1 or -3 to view the lockword.

We use `!!` instead of just `!` so that each file in the fileset being %ALTFILEd gets its own lockword — if we use one `!`, all the files being %ALTFILEd will get the same random lockword. This is because a single `!` would be resolved immediately at the time the %ALTFILE command is parsed, rather than just once for each file — see the chapter on [MPEX variables](#) for more information.

Security considerations

A file's lockword may be altered only by the account manager of the file's account or by the system manager.

Related MPEX features

You can also (if you have SM or AM in your own account) use MPEX selection criteria to find files by lockword (or the absence of one). For instance, you might say

```
%LISTF @.@.(LOCKWORD<>""),2
```

to find all the files in the system that have a lockword; or, if there are several files that you want to have the same lockword (e.g. some secure data files that only authorized people can access) and you want to periodically change all these lockwords, you can say

```
%ALTFILE @.@.(LOCKWORD="OLDLOCK"); LOCKWORD=NEWLOCK
```

and all the old `OLDLOCK` lockwords will be automatically changed to `NEWLOCK`.

Changing program file maxdata value (;MAXDATA=)

Syntax:

```
%ALTFILE fileset;MAXDATA=progmaxdata
```

Examples:

```
%ALTFILE AP010P; MAXDATA=30000
%ALTFILE @.PUB(ISPROG); MAXDATA=30000
```

The `;MAXDATA=...` keyword of the `:PREP` and `:RUN` commands indicates how much stack space a program can use. If your `MAXDATA` is too low, you're more likely to get Stack Overflows; some of these can be fixed just by running the program with a bigger `;MAXDATA=`.

`%ALTFILE...;MAXDATA=...` lets you change a program's `MAXDATA` value just as if you re-`:PREP`d with the new `MAXDATA`. This way, you neither have to re-`:PREP` nor always remember to specify a `;MAXDATA=...` on the `:RUN` command — just do an

```
%ALTFILE AP010P; MAXDATA=30000
```

(30000 is close to the maximum valid `MAXDATA` of 31232, but not so close that you might get into trouble with MPE running out of room in your stack to put its file system tables.)

Related MPEX features

In addition to letting you change a program file's MAXDATA, we also let you easily see the MAXDATA:

```
%LISTF MYPROG.DEV,5
FILENAME  MAXDATA  STACK  DLSIZE  DBSIZE  #SEGS  CAPABILITIES
MYPROG      30000   2048      0    4604    28  BA IA PM DS PH
```

(As you can see, all the various program parameters are shown.)

Actually, we're so flexible that you can even select based on MAXDATA (%LISTF @.@.(PROG.MAXDATA>30000),5).

Setting file flimits = to their eofs (;SQUEEZE)

Syntax: `%ALTFILE fileset;SQUEEZE`

Examples: `%ALTFILE RPT@.DATA; SQUEEZE`
`%ALTFILE @.DATA(MODDATE<TODAY-365); SQUEEZE`

Note: You probably want to use the ...;XLTRIM option instead of the ...;SQUEEZE option — it saves as much disk space, without restricting the further growth of the file.

Changing program file stack value (;STACK=)

Syntax: `%ALTFILE fileset;STACK=progstacksize`

Examples: `%ALTFILE AP010P; STACK=3000`
`%ALTFILE @.PUB(ISPROG); STACK=3000`

One of the parameters that you can specify on the :PREP= command is ;STACK=..., the initial Qi-Z area to be allocated for the process. This is a very little-known and very little-used feature; if you do use it, %ALTFILE...;STACK=... will help you change it if it was specified incorrectly on your :PREP command. In reality, this parameter is mostly included just for completeness. The really useful program file modification options are ;CAP= and ;MAXDATA=.

Releasing file unused disk space on MPE/iX (;XLTRIM)

Syntax: `%ALTFILE fileset;XLTRIM`

Examples: `%ALTFILE RPT@.DATA;XLTRIM`
`%ALTFILE @.@.(SAVABLESECTORS>0);XLTRIM; KEEPAMDATES`

MPE/iX files — especially those whose EOFs are less than their FLIMITs — often use more space than necessary. This happens because MPE/iX doesn't want to allocate disk space a record at a time; this would take too long and cause much disk fragmentation. MPE/iX allocates space in units of up to 2048 sectors, so it's possible that a 1-record file uses an entire 2048-sector chunk of (contiguous!) disk space.

The %ALTFILE...;XLTRIM command lets you release all this unused space without changing the file's file limit. Thus, you can save disk space without making the file any less extensible — when you try to append more data to the file later, MPE/iX will just allocate another chunk of space.

%ALTFILE...;XLTRIM may increase file fragmentation (for files that you frequently append new records to) — instead of being a collection of 2048-sector chunks, a file that is repeatedly XLTRIMed becomes a larger collection of smaller chunks. We don't think that this should be much of a problem, but you might want to keep it in mind.

Our experience has been that you can save truly dramatic quantities of disk space this way without making the XLTRIMed files any less extensible (since their file limits remain unchanged). You might also want to use the %LISTF..., SAVABLE command to get an idea of just how much disk space can be saved this way.

Changing the internal structure of a file

Syntax:

```
%ALTFILE filespec;FIXED
%ALTFILE filespec;VARIABLE [;KEEPTRAIL]
%ALTFILE filespec;UNDEF
%ALTFILE filespec;BYTE
%ALTFILE filespec;ASCII
%ALTFILE filespec;BINARY
%ALTFILE filespec;REC=maxrecordlength ;VARIABLE [;KEEPTRAIL]
```

Examples:

```
%ALTFILE RPT@.DATA;ASCII;VARIABLE;REC=80;KEEPTRAIL
%ALTFILE NEW.UPLOAD;BINARY
%ALTFILE ./doc/chapter2;BYTE
```

The keywords FIXED, VARIABLE, UNDEF and BYTE allow you to change the internal structure of a file during an %ALTFILE or %COPY operation. Note that these keywords are mutually exclusive, that is, you may specify only one of these keywords. The same is true for ;BINARY and ;ASCII; however, either of these may be combined with one of the previous keywords (i.e. you can specify ";FIXED;ASCII" or ";BINARY;VARIABLE", but not ";VARIABLE;ASCII;BINARY" or ";BINARY;UNDEF;BYTE").

When converting files to FIXED or VARIABLE length, you have the option of specifying the new fixed length or the maximum length using the ;REC= keyword.

Files that are converted to FIXED length will have trailing spaces (if ASCII) or NULLS (for BINARY) appended to each record. Files that are converted to VARIABLE ASCII will have trailing spaces REMOVED unless the ;KEEPTRAIL keyword is specified.

%ALTJOB

Syntax: `%ALTJOB userset; altjobparms
[;NOVERIFY]
[;QUIET]`

Examples: `%ALTJOB @.AP&WAIT; INPRI=4
%ALTJOB WAIT; OUTDEV=LP2
%ALTJOB REPORTJ,@.@; INPRI=12`

The %ALTJOB command is an enhanced version of MPE's :ALTJOB command. It allows you to specify which jobs to alter using VESOFT usersets (see the Usersets section of the SECURITY User Manual).

The ;NOVERIFY and ;QUIET parameters are the same as those used in the %ABORTJOB command. For more information, see the documentation for the [%ABORTJOB command](#).

%ALTPROC

Syntax: `%ALTPROC [[PIN=] [#P]pin | [PIN=](pin [,...])]
[[JOB=] userset | JOB=(jobID [,...])]
[;PRI= { CS|DS|ES }]
[;[NO]TREE]
[;{USER|ANYUSER}]
[;SYSTEM]`

Examples: `%ALTPROC #J232;PRI=CS
%ALTPROC PIN=100;PRI=DS
%ALTPROC BATCH-#J200;PRI=ES`

Note: This command is only available on MPE/iX systems.

The %ALTPROC command is used to alter the priority of a process or processes. MPEX's %ALTPROC command is an improved version of the MPE/iX :ALTPROC command.

The main enhancement is that it supports VESOFT usersets (see the Usersets section of the Security User Manual for details) in the job specification. Please refer to HP's documentation for an explanation of the various options.

%ALTSCHED

Syntax: `%ALTSCHED scheduledjobset [schedparms]
[;NOVERIFY]
[;QUIET]`

Examples:

```
%ALTSCHED BACKUP,MANAGER.SYS; AT=23:00
%ALTSCHED @.AP; IN=0,1,0
%ALTSCHED MONDAY,PR.PROD; DAY=TUESDAY
%ALTSCHED #J23; DATE=12/31/90
%ALTSCHED MYJOB,USER.DEV;NOVERIFY
```

Note: This command is only available on MPE/iX systems. MPE/V release 3P contains support for altering the submission time of jobs in the SCHED and WAIT queues.

The %ALTSCHED command lets an SM user change the scheduling parameters of an existing job in the SCHED state. You can specify which jobs to re-schedule using VESOFT usersets (see the Usersets section of the SECURITY User Manual for details) and any MPE :STREAM scheduling parameters (;AT=, ;DAY=, ;DATE=, ;IN=).

This is especially useful if you need to re-schedule a job that you no longer have the source for or a STREAMX job using parameter substitution (see the SECURITY User Manual for details) and you don't know what parameters it was :STREAMed with.

For example, if you have a job scheduled to run at 11:00 PM and decide you need it to run at 10:00 PM instead, the command

```
%ALTSCHED #J1234; AT=22:00
```

will re-schedule it for you.

Note that after a job has been re-scheduled by %ALTSCHED, it will have a new job number.

Another use of %ALTSCHED is to immediately submit a scheduled job. Since we use normal MPE syntax for the new scheduling parameters, entering "%ALTSCHED *jobset*" without any scheduling information will default to "stream it NOW".

By default, unless you specify a single particular job number, ALTSCHED will show the job numbers and logon IDs of all the jobs that are to be re-scheduled and then ask you if it's OK to process them. If you do not reply "Y", the jobs will not be re-scheduled. This can help you avoid accidentally altering many jobs that you don't actually want to alter.

If you want to do the %ALTSCHED without prompting for verifications, just append ";NOVERIFY" to the command, e.g.

```
%ALTSCHED @,MANAGER.SYS; AT=23:45; NOVERIFY
```

This will still show the numbers and logon IDs of the jobs that are being processed; using the ";QUIET" keyword, e.g.

```
%ALTSCHED @.AP; IN=0,1,0; QUIET
```

will turn off both the verification request and the display of the job numbers and logon IDs.

As mentioned above, if you specify a single job/session number (e.g. "%ALTSCHED #J1234...") you will not be asked for verification, and the job number/logon will not be displayed, just as if you had said ";QUIET".

Note: This command cannot alter "#A" type jobs (jobs streamed by the STREAMX scheduler).

%ALTSEC

Syntax: `%ALTSEC fileset; [altsecparms]`

Examples:

```
%ALTSEC PERS@.MGRMEMOS; (R,W,A,L,X:CR)
%ALTSEC @.@.TELESUP(PROG.PMCAP); (R,X:AC;W,A,L:CR)
%ALTSEC MYDB+MYDB?#; (R,W,A,L,X:AL,GU)
%ALTSEC @.DATA(NOT HASACD); NEWACD=(R:@.PROD;W:MGR.PROD)
%ALTSEC @.@.DEV; COPYACD=AP010.PUB.DEV
```

The %ALTSEC command lets you alter the security characteristics — access masks and ACDs — of entire FILESETS.

Particularly useful applications include:

- One of the troubles with :ALTSEC is that whenever you re-build a file (e.g. using :EDITOR's /KEEP, which purges the target file and re-builds it), the security matrix is set back to READ, WRITE, APPEND, EXECUTE, LOCK: ANY, and any ACDs set up for the file are lost! Therefore, it's a good idea to regularly %ALTSEC filesets that you know should be specially protected, just in case somebody re-built one of those files and forgot to manually re-:ALTSEC it. For example:

```
%ALTSEC PERS@.MGRMEMOS; (R,W,A,L,X:CR)
```

or

```
%ALTSEC PERS@.MGRMEMOS; NEWACD=(R,W:JOHN.MGRMEMOS)
```

- MPEX's powerful selection criteria can be used to intelligently control security on specific files. For instance:

```
%ALTSEC @.@.TELESUP(PROG.PMCAP); (R,X:AC;W,A,L:CR)
```

will %ALTSEC all the privileged program files (PROG.PMCAP) in the TELESUP account to prevent anybody other than TELESUP users (and users with SM capability) from running them.

Similarly, if you use ACDs, you can say

```
%ALTSEC @.DATA(NOT HASACD and ISASCII); NEWACD=(R:@.PROD;W:MGR.PROD)
```

to alter all the ASCII files (ISASCII) in the DATA group that don't already have an ACD.

- Finally, the %ALTSEC command works on databases (DBUTIL has >>RELEASE and >>SECURE commands, but no >>ALTSEC command).

Saying

```
%ALTSEC MYDB+MYDB?#; (R,X,W,A,L:AL,GU)
```

will alter the MYDB database so that only users in the same group or users with AL capability can access it. Note, however, that only the access mask (not the ACD) features of %ALTSEC can be used on databases – MPE doesn't let us define ACDs for databases.

However, the VEOPEN module of VESoft's SECURITY product lets you restrict access to database not just by user ID (as ACDs would) but also by program, by session name, by mode, and by user class! See Database Security in the SECURITY User Manual for details.

An additional advantage of MPEX's %ALTSEC is that it does not require creator access when it is used to alter a file's access mask. It can be used by anybody with Read, Write, and eXecute access to a file (since anyone who has Read, Write, and eXecute access can already do anything to the file); this can be very useful when, say, the System Manager (or an account manager) wants to %ALTSEC a whole fileset of files that may be in many different accounts and may have been created by many different users.

The %ALTSEC alter-ACD options can be used only by a user with SM capability, a user with AM altering files in his own account or the file's creator — this is exactly the same as the requirements of MPE's alter-ACD options.

%ALTSPoolFILE

Syntax:

```
%ALTSPoolFILE spoolfileset; {altspoolfileparms|spoolfparms} [ ;... ]
```

Examples:

```
%ALTSPoolFILE $STDLIST.@.(SPOOL.JSNAME="MCOMPILE"); PRI=3
%ALTSPoolFILE RPTLIST.DEV.AP(SPOOL.NUMCOPIES=5); COPIES=1
%ALTSPoolFILE @.@.(SPOOL.DEVICE="FASTLP"); DEV=LP; PRI=1
%ALTSPoolFILE REPORT.PUB.PR; UNDEFER (MPE/iX)
```

The %ALTSPoolFILE command is just like MPE's :ALTSPoolFILE (and MPE/iX's :SPOOLF command), except that:

- It supports spool file filesets, letting you alter many spool files at once, WITHOUT having to know their #0xxx numbers.
- It can be performed not just by the console operator, but also by:
 - anybody with SM or OP capability,
 - anybody with AM capability working on spool files in his own account,
 - anybody working on his own spool files.

(Note that this is the same security system used by SPOOK5.)

As the examples above show, this can be an exceptionally powerful command, not just because you can perform it on several spool files at once, but also because you can indicate the spool file not by its (completely arbitrary) #0xxx number, but by its spool file name, creator job name, priority, etc.

For instance,

```
%ALTSPoolFILE $STDLIST.@.(SPOOL.JSNAME="MCOMPILE"); PRI=3
```

will find all the job \$STDLISTs created by jobs with job name MCOMPILE and alter their output priority to 3.

MPE/iX native mode spooler note:

The Native Mode Spooler on MPE/iX does not retain the spoolfile's group name. If you specify a group name in %ALTSPoolFILE on MPE/iX, it will be effectively ignored. For example:

```
%ALTSPoolFILE $STDLIST.PUB.PROD...
```

is the same as

```
%ALTSPoolFILE $STDLIST.@.PROD...
```

%BACKG

Syntax:

```
%BACKG      [SHOW]
              [STARTJOB]
              [STOPJOB]
              [{START|STOP}, taskname]

"taskname" is one of:

              ALARM    AUDITC    CMDPROT    HELLO    LOGOFF
              LOGON    NETPUSH    NETRECV    OBSFILL

or the name of a user-implemented task.
```

Examples:

```
%BACKG STARTJOB
%BACKG START,LOGOFF
%BACKG STOP,LOGOFF
%BACKG SHOW
%BACKG STOPJOB
```

A number of MPEX, SECURITY, and VEAUDIT features require a background task that either runs continuously (ALARM, CMDPROT, HELLO, LOGOFF, NETRECV) or periodically (NETPUSH, OBSFILL, AUDITC). In order to help you better manage your system, VESoft provides a special facility that will execute all of these tasks within a single background job!

Note: Not all tasks shown here are available on all machines. For example, the HELLO and LOGON tasks make use of an operating system feature that is unavailable on CLASSIC systems and pre-4.0 XL systems.

The BACKG background job is controlled by an interactive command, which you can enter through MPEX, SECURITY or VEAUDIT (in these examples only, what you type is underlined):

```
:RUN MPEX.PUB.VESoft
%BACKG
```

or

```
:RUN LOGON.PUB.VESOFT, SEC
%SEC BACKG
```

or

```
:RUN VEAUDIT.PUB.VESOFT, VEAUDITCMD
%VEAUDIT BACKG
```

For the sake of brevity, all examples will use MPEX; just remember that you can use SECURITY or VEAUDIT for any of the BACKG commands.

The %BACKG command has options to start and stop the background job itself, and to start, stop, and show the individual tasks being run by the background job. (If you type %BACKG without any parameters, it does a SHOW.)

%BACKG STARTJOB (requires SM or OP capability)

The easiest way to start the background job is to type:

```
%BACKG STARTJOB
```

from the MPEX prompt. MPEX will submit the job using the same technique that STREAMX uses to insert passwords for MANAGER.VESOFT, so we require the user starting the job to have SM or OP capability. MPEX checks whether the background job is already running before actually submitting the job.

Note that if you use the %ALARM command (documented in this manual) to set an alarm and the BACKG job is not running, it will be started for you automatically, but with only the ALARM task active. If %BACKG STARTJOB is subsequently issued, any other pending BACKG tasks will be started at that time.

%BACKG STOPJOB

To stop the background job, type

```
%BACKG STOPJOB
```

or simply use the :ABORTJOB command.

Note: If you stop the background job using %BACKG STOPJOB without first stopping every individual task (using %BACKG STOP, *task* documented below) then each task will be automatically re-started the next time you re-start the background job. This makes it very convenient to be able to restart the BACKG job after your system backup without having to remember which tasks were active at the time.

Stopping the BACKG job for a backup is important because several of the tasks may have control files or the SECURG file open while they run. If these files are in use, then the :STORE command may not store these files to tape, which may subsequently cause problems if you need to recover your system using these tapes.

%BACKG START, *taskname*

This command tells the background job to begin execution of one of the tasks under its control. Some of the tasks will begin execution immediately, and continue until you stop them (or stop the entire job). Others will suspend until their scheduled execution time, when they will execute once and then suspend again until the next day. We'll describe later exactly which ones do what, and how you can control when the intermittent tasks are to execute.

Here are a few examples:

```
%BACKG START,LOGOFF      (* to start LOGOFF *)
%BACKG START,OBSFILL     (* OBSFILL will run once a day *)
```

Note: Because all of the tasks run in a single job, we re-direct the `STDLIST` for each task to a file (named *S taskname*) in the `DATA` group (e.g. `SLOGOFF`, `SOBSFILL`, etc.) — otherwise, reading the `$STDLIST` of the `BACKG` job, you wouldn't be able to tell which lines were printed by which tasks.

Each of these files is overwritten every time you restart the corresponding `BACKG` task.

%BACKG STOP, *taskname*

This command tells the background job to stop executing the specified task. Examples:

```
%BACKG STOP,CMDPROT
%BACKG STOP,AUDITC
```

Note: If you stop the background job but don't use `%BACKG STOP` to terminate a particular `BACKG` task first, that task will be automatically re-started the next time you re-start the background job. For instance, if you have `LOGOFF` running, and you stop the background job (by doing a `%BACKG STOPJOB`, `:ABORTJOB`, etc.), `LOGOFF` will be stopped; however, the next time you start the background job (e.g. by doing a `%BACKG STARTJOB`), `LOGOFF` will be re-started automatically.

%BACKG SHOW

This command shows you the current status of all available background tasks: either "Initializing", "Executing", "Not Executing" or "Aborted". If the background job is not running, all tasks will show as either "Not Executing" or "Pending".

In the case of intermittent tasks like `OBSFILL` that are usually "sleeping" (waiting to execute at a future time) "Executing" means they will execute at some future time; "Not Executing" means they will not execute.

```
%BACKG SHOW,taskname
```

This form of the `%BACKG SHOW` command will display the `$STDLIST` of a particular task, whether or not the task is currently active. The `$STDLIST` is stored in the file '*S taskname*' in the `DATA` group.

BACKGINI.DATA configuration file

Whenever you start the BACKG job, the first thing it does (before it starts any of the background tasks) is to execute any commands in the file BACKGINI.DATA.

This can be used, for example, to issue a file equation for LOGOFF.DATA.VESOFT or to specify when the intermittent tasks (ALARM, NETPUSH, OBSFILL, AUDITC) are to run by setting special variables (described below).

If you make any changes to BACKGINI.DATA, you must stop and then re-start the BACKG job before the changes will take effect.

Setting execution priority for BACKG tasks

If you want to have any of the background tasks execute at a higher (or lower) priority than the default you have set for batch jobs, all you have to do is add a

```
SETVAR tasknamePRI "xS"
```

(where "*taskname*" is one of the background task names, and "xS" is the subqueue [CS, DS or ES] that you want that task to execute in) to your BACKGINI.DATA file. For example, if you want LOGOFF to run in the CS queue, you would add to your BACKGINI.DATA.VESOFT file the line

```
SETVAR LOGOFFPRI "CS"
```

BACKG tasks

The file TASKLIST.BACKG.VESOFT contains a list of valid task names. Additionally, if you decide that you do not want to use a particular task, you can remove it from this list and it will be disabled.

Here are the tasks that can be run using %BACKG, plus notes on any configuration options for them that you can put in BACKGINI.DATA:

ALARM This task implements the MPEX "alarm" facility (see the [%ALARM command](#) in this manual).

By default, the ALARM task wakes up once per minute; you can change this (to, say, 5 minutes) by adding to your BACKGINI.DATA file:

```
SETVAR VESOFTALARMPAUSE 300
```

Note: The ALARM task is used by the STREAMX SCHEDULER to schedule repeating or conditional jobstreams. If you intend to use this feature, the ALARM task must be active.

AUDITC This task collects the information VEAUDIT needs to produce the reports on "MPE Passwords Unchanged In The Last 30 Days", "SECURITY Passwords Unchanged In The Last 30 Days", "Recent MPE Directory Changes", and "Recent SECURITY Users Changes". AUDITC should be run daily.

By default (if you've done a %BACKG START,AUDITC), it will be run at 2AM. To change this to 3:15AM, add the following line to BACKGINI.DATA.VESOFT:

```
SETVAR VEAUDITCTIME "3:15AM"
```

CMDPROT This task implements the VEAUDIT facility which protects your UDC catalog, **COMMAND.PUB.SYS**. It is described in the VEAUDIT User Manual under "Section #105: Users Who May Disable System UDCs".

It runs continuously, but in a "MSG WAIT" state so it doesn't use any CPU time after starting up.

HELLO This task makes use of a Procedure Exits (PE) to automatically intercept and authorize network logons. Starting this task enables the PE, and stopping the task disables the PE.

Note that this task must run continuously to trap logons from such products as ARPA and FTP which circumvent all UDCs (including any that use **OPTION LOGON**).

See the SECURITY manual 'NETWORK SECURITY...' chapter for details.

Note: As Procedure Exits are used, this task is only available on MPE/iX 4.0 and later systems.

LOGOFF This task implements SECURITY's facility for aborting idle sessions (a real security risk — see "Unattended Terminal Security" in the SECURITY User Manual).

By default, LOGOFF uses the files **LOGOFF.DATA**, **LOGOFFAB.PUB.VESOFT**, and **LOGOFFWR.PUB.VESOFT**; you may put file equations for any or all of these files in **BACKGINI.DATA.VESOFT**. LOGOFF generally wakes up every five minutes, but its operation is controlled by the file **LOGOFF.DATA.VESOFT**.

LOGON This task enforces SECURITY/3000 logon checking regardless of whether or not the UDC file **LOGONUDC.PUB.VESOFT** has been **:SETCATALOGed** (or ignored by SM users using the **;PARM=-1** option of the **:HELLO** command).

This task makes use of a Procedure Exit (PE) that traps all logons and runs our LOGON program. Starting this task enables the PE which will then continue to trap logons even if the BACKG job is aborted by the **ABORTJOB** command. Stopping the task with the **%BACKG STOP** command will disable the trap and return your system to normal.

Note: As Procedure Exits are used, this task is only available on MPE/iX 4.0 and later systems.

NETRECV This task is part of the multi-CPU user profile maintenance facility of SECURITY (see the SECURITY User Manual). It runs continuously, but only uses CPU when there is a user profile change.

NETPUSH This task is part of the multi-CPU user profile maintenance facility of SECURITY.

NETPUSH runs once a day to forward SECURITY user profile changes to other systems. By default, %BACKG will run NETPUSH every day at 1AM (but only if you've entered a **%BACKG START, NETPUSH**). If you want NETPUSH to run at 2:30 AM, add the following line to **BACKGINI.DATA.VESOFT**:

```
SETVAR NETPUSHTIME "2:30AM"
```

OBSFILL This task is part of SECURITY's MPE-password obsolescence facility; it collects all MPE password changes. OBSFILL runs once a day to see if any MPE passwords have been changed during the day. By default (if you've done a %BACKG START ,OBSFILL) it will be run at 1AM.

As above, if you want to run it at a different time simply add a line to BACKGINI.DATA.VESOFT, e.g.:

```
SETVAR OBSFILLTIME "11PM"
```

%BREAKJOB

Syntax:

```
%BREAKJOB userset
           [;NOVERIFY]
           [;QUIET]
```

Examples:

```
%BREAKJOB @.AP
%BREAKJOB @.@
%BREAKJOB REPORTJ,@.@
%BREAKJOB #J1234+#J1235
```

The %BREAKJOB command is an enhanced version of MPE's :BREAKJOB command. It allows you to specify which jobs to suspend using VESOFT usersets (see the Usersets section of the SECURITY User Manual). Of course, this command will only apply to jobs that are in the EXEC state. It uses the same parameters as %ABORTJOB. For more information, see the documentation for the [%ABORTJOB command](#).

%BYE

Syntax:

```
%BYE [BYE|QUIT|ASK]
```

Examples:

```
%BYE
CPU=5.  Connect=1.  WED, DEC 13, 1995,  4:09 PM.
```

```
%BYE QUIT
END OF PROGRAM

%BYE ASK
OK to logoff (y/N)? n
END OF PROGRAM
```

The %BYE command is a modified version of MPE's :BYE command. It allows you to terminate the current MPEX process, and also (on MPE/iX systems) to terminate the current Session. Like MPE's :BYE command, %BYE may not be used within Jobs.

MPEX COMMAND REFERENCE: %CALC

By default (or when the `;BYE` keyword is specified), you will be logged off when `%BYE` is executed. If you specify the `;QUIT` keyword, MPEX will terminate, but the active session will not be logged off. If you specify the `;ASK` keyword, then you will be prompted whether to log off or not.

`%BYE` always checks to see if MPEX has any active son processes (perhaps as a result of using the `%GOON` command), and will fail if any active son processes exist. You must either wait for the active sons to terminate or use the `%KILL` command to terminate them before `%BYE` will succeed.

The `VESOFTDEFAULTBYE` variable may be used to change the default behavior of the `%BYE` command.

To ensure that `%BYE` will terminate MPEX without also logging users off the system, add the following line to your `MPEXMGR.PUB.VESOFT` file:

```
%SETVAR VESOFTDEFAULTBYE "QUIT"
```

To change `%BYE` so that users will be prompted whether they wish to logoff or not, instead add the following line:

```
%SETVAR VESOFTDEFAULTBYE "ASK"
```

The `;BYE` keyword allows you to override any `VESOFTDEFAULTBYE` setting, and guarantees that your session will be terminated.

%CALC

Syntax:

```
%CALC expression
```

Examples:

```
%CALC 123*456+789
56877, $DE2D, %157055, "...-"
    << output in decimal, hexadecimal, octal, and ASCII >>
    << ASCII has unprintable chars replaced by "." >>
%CALC 456.0//5.8
78.620689                << real arithmetic >>
%CALC TODAY+5
93/02/04                  << date arithmetic >>
```

```
%CALC STR("TESTING"+"ONE"+"TWO"+"THREE",6,12)
NGONETWOTHRE             << string manipulation >>
%CALC MPEXNUMSUCCEEDED/(MPEXNUMFAILED+MPEXNUMSUCCEEDED)
0.950000
```

The `%CALC` command evaluates an arbitrary expression (see [Appendix B](#) for details on expression syntax and the available operators).

The expression can involve integers, strings, logical values, real numbers, dates, and times; it can also operate on MPEX VARIABLES (see the [MPEX variables](#) chapter) as well as on constants.

If the expression has an integer result, the result will be displayed in decimal, hexadecimal, octal, ASCII representation of the 4 bytes, with "garbage" (i.e. unprintable) characters represented by periods (".").

In addition to displaying the result, %CALC also sets the HPRESULT variable to the calculated value.

%CALENDAR

Syntax:

```
[monthname]
%CALENDAR [month] [,year]
           [ @ ]
           [ ;NOHOLIDAY ]
           [ ;PICTURE ]
```

Examples:

```
%CALENDAR FEB
%CALENDAR OCTOBER,2004
%CALENDAR 2,68;NOHOLIDAY
%CALENDAR @;PICTURE
```

Prints a calendar for the given month (or for the whole year if month is given as "@"). The month can be given as a number (e.g. 10) or as a string (e.g. OCTOBER), optionally abbreviated to the first three letters (e.g. OCT). The year may be entered as a two or four digit year. The default month is the current month; default year is the current year. If you're printing the calendar for the current month, then today's date will be highlighted.

Certain hard-to-remember holidays are automatically listed after the calendar.

(You can change which holidays are listed by creating your own files similar to the HOLID##.DATA.VESOF (HOLID01 through HOLID12) and issuing file equations for them.)

%CHGROUP

Syntax:

```
%CHGROUP [groupname [/grouppass]]
           [ ;KEEPCAPS ]
           [ ;KEEPALLOW ]
           [ ;KEEPUDCS ]
```

Examples:

```
%CHGROUP DATA
%CHGROUP PUB; KEEPCAPS
%CHGROUP
```

The %CHGROUP command (as you can probably guess) allows you to switch to a different group. If you don't specify which group to change to, you will be changed to your home group. If the new group has a group password (and you don't include it in the %CHGROUP command), you will be prompted for it unless you have SM or AM capability or you are changing to your home group.

MPEX has a function called `ISCHGROUPED()` that returns TRUE if the user has issued an MPEX `%CHGROUP` command and is currently in a different group to the logon group. After the `%CHGROUP` command is used to return to the original logon group, `ISCHGROUPED()` returns FALSE. This is more specific than the `ISCHLOGONED()` function, which returns TRUE if the user has changed logons directly (using `%CHLOGON`) or indirectly (using `%CHGROUP`). In fact, using `%CHLOGON` to change only the group will cause `ISCHGROUPED()` to return TRUE (which is actually what `%CHGROUP` does internally).

One thing to be aware of with `CHGROUP` (and `CHLOGON`, for that matter) is that the predefined variable `HPGROUP` will change to reflect the new group name.

What this means is that if you have:

```
SETVAR MPEXPROMPT "!HPGROUP.!HPACCOUNT: "
```

or anything similar in your MPEXMGR start-up file, you should change it to:

```
SETVAR MPEXPROMPT "!!HPGROUP!!HPACCOUNT: "
```

so that when you issue the `%CHGROUP` command your prompt changes (otherwise your `MPEXPROMPT` variable will remain set to the same value even though you changed from one group to another — see the discussion of "!"s with regard to [variable substitution](#)).

`%CHGROUP` is really just a subset of the more general `%CHLOGON` command, documented later in this manual (please see the [%CHLOGON command](#) for an explanation of the various parameters). In particular, `%CHGROUP` is affected by `$CHLOGON-NOPASS`, `$CHLOGON-FORBID`, and `$CHLOGON-PERMIT` in `STREAMX.DATA` and any appropriate `$LOGON-EXECUTE` commands in `SECURCON.DATA` (just as if you had entered a `%CHLOGON` command).

Also, just like with `%CHLOGON`, we must disable the `[BREAK]` key when you do a `%CHGROUP` until you do a `%CHLOGON` with no parameters; simply entering a `%CHGROUP` with no parameters will not re-enable break if you didn't originally log on to your home group; `%CHGROUP` with no parameters switches you to your home group (for compatibility with MPE/iX); `%CHLOGON` with no parameters switches you back to your original logon. Please read the ["Important note for MPE/iX users"](#) in the `%CHLOGON` documentation.

%CHLOGON

Syntax:

```
%CHLOGON [[session,] user [/pass] .account [/pass] [,group [/pass]]]
           [;KEEPCAPS]
           [;KEEPALLOW]
           [;KEEPUDCS]
           [;SILENT]
%CHLOGON abbreviated-logon
```

Examples:

```
%CHLOGON EUGENE,MANAGER.SYS
%CHLOGON =,CLERK.PR;KEEPCAPS;KEEPALLOW;KEEPUDCS;SILENT
%CHLOGON
```

Special note: Although this feature is being described in the MPEX User Manual, some of the features used to control it (the \$CHLOGON-xxx commands in the STREAMX.DATA file), are only available to you if you are a user of both MPEX and SECURITY. This command is also available (to SECURITY users) within STREAMX jobs (via ::CHLOGON) and in SECURITY menus.

The %CHLOGON command gives you the ability to switch to a different MPE account, group, user or session name without having to re-logon via the :HELLO command!

Why not just re-logon via :HELLO?

- The :HELLO command creates an entirely new session. You lose all of your file equations, variable settings, temporary files, REDO history, etc.

%CHLOGON preserves all of this (and more!) for you.

- Because it doesn't have to actually create an entire new session, %CHLOGON is much faster than :HELLO.
- %CHLOGON can be used in command files, SECURITY menus, and in STREAMX as " :CHLOGON"!
- %CHLOGON lets you KEEP all of the CAPabilities (;KEEPCAPS), ALLOWs (;KEEPALLOW), and UDCs (;KEEPUDCS) from your original logon in your new logon; this includes extra capabilities and ALLOWs acquired via the GOD program or SECURITY's \$ALLOW mechanism.
- Like STREAMX, %CHLOGON can be configured to not prompt for MPE and SECURITY user profile passwords. This means you can set up a command file or menu (with OPTION NOBREAK, and to which the user doesn't even need read access, only execute) that logs on to another account, performs some task, then switches the user back to his original logon. This is much more secure than giving him the passwords to an account he doesn't normally need to log on to.

As you can see, %CHLOGON (:CHLOGON in STREAMX, and CHLOGON in menus) uses the same basic syntax as MPE's :HELLO command, but with a few special keywords added and the "user.account" is optional — if you don't specify a "user.account", %CHLOGON simply switches you back to your original logon, the one you entered at the :HELLO command. This is especially useful in command files, SECURITY logon menus, and STREAMX jobs: you can do a "CHLOGON newuser.newacct", perform whatever task you want under the new logon, then do a "CHLOGON" with no parameters to switch back to the original logon.

You may use an "=" in place of the session, user, account, and/or group names to retain your current session, user, account, and/or group.

After you enter the %CHLOGON command, you will be prompted for the appropriate MPE and SECURITY password(s) (unless you have SM or AM and are changing to another logon in the same account or there is a \$CHLOGON-NOPASS – documented below — in effect for you), and then you will be switched to the new logon.

One thing to be aware of with CHLOGON (and CHGROUP for that matter) is that the redefined variables HPGROUP and HPACCOUNT will change to reflect the new logon ID.

What this means is that if you have:

```
SETVAR MPEXPROMPT "!HPGROUP.!HPACCOUNT: "
```

or anything similar in your MPEXMGR start-up file, you should change it to:

```
SETVAR MPEXPROMPT "!!HPGROUP.!!HPACCOUNT: "
```

so that when you issue the %CHLOGON command your prompt changes (otherwise your MPEXPROMPT variable will remain set to the same value even though you changed from one logon ID to another — see the discussion of "!"s with regard to [variable substitution](#)).

The following special keywords may be used in the %CHLOGON command:

- **;KEEPCAPS** tells %CHLOGON to give you the same capabilities under your new logon that you had under your old one (this includes any capabilities acquired via the **GOD** program).
To prevent AM users in one account from getting AM in another account (where they might not normally have it), **;KEEPCAPS** only works if you have SM capability.
- **;KEEPALLOW** tells %CHLOGON to **ALLOW** you the same console commands that you were **ALLOWed** under your old logon (this includes global **ALLOWs**, **:ALLOWs** issued by the console operator, **ALLOWs** acquired via the **GOD** program, and **\$ALLOWs** in your **SECURITY SECURCON.DATA** file).
- **;KEEPUDCS** tells %CHLOGON to give you the same UDCs under your new logon that you had under your old one.
- **;SILENT** tells %CHLOGON to switch to the new logon without displaying the message "Welcome! You are now signed on". This is useful when you want to use %CHLOGON within a command file or menu where the user does not need to know that you changed their logon-ID.

The function **ISCHLOGONED()** will return **TRUE** if the current session has executed a %CHLOGON (and has not yet switched back). This can be useful within a command file or **SECURITY** menu to test the **CHLOGON** status. In addition, the **ISCHGROUPED()** function returns **FALSE** if the %CHLOGON command changes the account or user, but **TRUE** if **ONLY** the group changed (which is actually what %CHGROUP does internally).

Eliminating password prompts (\$CHLOGON-NOPASS)

You can allow particular usersets, using particular MPEX command files/SECURITY menus/STREAMX jobs, to switch to particular logons without being prompted for passwords by adding entries of the form:

```
$CHLOGON-NOPASS currentuserset fileset targetuserset
```

to your **STREAMX.DATA** file (as you can see, this is quite similar to the **\$NOPASS** and **\$WITHCAPS-PERMIT/FORBID** entries in **STREAMX.DATA**). For example:

```
$CHLOGON-NOPASS @.DEV TESTPROD.CMD.PROD TEST.PROD
```

means anyone in the **DEV** account can %CHLOGON to **TEST.PROD** with **ANY** session name by using the command file **TESTPROD.CMD.PROD**.

The "*fileset*" above can include \$STDIN; this is how you keep from asking for passwords when the user enters "%CHLOGON..." directly from the MPEX "%" prompt.

Restricting who may use %CHLOGON (\$CHLOGON-FORBID|PERMIT)

What if you don't want certain users to have access to %CHLOGON at all? There are two keywords you can put in the STREAMX.DATA file to control who can use %CHLOGON:

```
$CHLOGON-FORBID  currentuserset  fileset  targetuserset
$CHLOGON-PERMIT  currentuserset  fileset  targetuserset
```

By default, all users are allowed to use %CHLOGON to switch to any logon that they know the passwords for. \$CHLOGON-FORBID lets you forbid a particular userset from switching to a particular set of new logons via a particular fileset of MPEX command files, SECURITY menus, and/or STREAMX jobs. This "*fileset*" can include (or exclude) \$STDIN; this is how you control use of "%CHLOGON..." in an interactive session. \$CHLOGON-PERMIT cancels the effect of a previous \$CHLOGON-FORBID. This lets you say things like

```
$CHLOGON-FORBID  @.PROD  @.@.@  @.@
$CHLOGON-PERMIT  BERT,@.PROD  @.MUPPET.PROD  ERNIE,@.PROD
```

which means no users in the PROD account may use %CHLOGON at all, except any user in PROD with a session name of "BERT" can use any file in the group MUPPET.PROD to %CHLOGON his session name to "ERNIE". Another example would be:

```
$CHLOGON-FORBID  @.@          @.@.@  @.@
$CHLOGON-PERMIT  @.DEV        $STDIN.@.@  @.@
```

Which means that only the users who can use the %CHLOGON command are users in the DEV account, and even then, they can only use it from a "%" prompt.

What if the new logon is protected by a SECURITY logon menu?

- If you have SM capability (and use ;KEEPCAPS), you will be switched to the new logon and the menu will not be activated;
- If you don't have SM capability, you will not be permitted to switch to the new logon at all.

If you want non-SM users to be able to switch to logons that are protected by SECURITY logon menus (bypassing the logon menu), add the keyword

```
$CHLOGON-OKMENU  currentuserset  fileset  targetuserset
```

to your STREAMX.DATA.VESOFT file. This keyword only allows a user to bypass the logon menu for the new logon if he knows the passwords (or if you also have a \$CHLOGON-NOPASS for him).

For example:

```
$CHLOGON-OKMENU  KENT,OPERATOR.SYS  BOOTH.CMD.SYS  BOSS,MANAGER.SYS
```

means "the user KENT,OPERATOR.SYS can use the command file BOOTH.CMD.SYS to CHLOGON to BOSS,MANAGER.SYS, even if BOSS,MANAGER.SYS is protected by a logon menu (the logon menu will be skipped)."

What if you want the user to be in the menu when he switches to the new account? Easy!
Using the same example as above, `BOOTH.CMD.SYS` could look something like this:

```
OPTION NOBREAK
CHLOGON BOSS,MANAGER.SYS
FILE MENUFILE=BOSS.MENU.SYS
RUN MAIN.PUB.VESOFT,MENU
CHLOGON
```

In addition to the `STREAMX.DATA` keywords of `$CHLOGON-NOPASS`, `$CHLOGON-FORBID`, and `$CHLOGON-PERMIT`, `%CHLOGON` (and `%CHGROUP`) will also execute `$LOGON-EXECUTE` commands from `SECURCON.DATA` as well. Please refer to the SECURITY manual discussion of `$LOGON-EXECUTE` for details on this keyword.

Using abbreviated logons with %CHLOGON

If you have configured abbreviated logons for the LOGON facility of our SECURITY package, then `%CHLOGON` will automatically recognize and use these abbreviated logons just as if you had typed the entire logon string manually. Abbreviated logons are only available when the `BACKG` job is running and the `HELLO` task is active. See the Additional Benefits of the `VESOFT HELLO` trap section of the SECURITY manual for details on abbreviated logons and the `HELLO` trap.

Important note for MPE/iX users

Due to MPE/iX limitations we cannot change your logon ID for other processes in your process tree, including your father process, other son processes of your father (brother processes), and any of your son processes that existed before you did the `%CHLOGON` (created, perhaps, by the `%GOON` or `%SPOONFEED`ing facilities or programs like `QEDIT` that suspend themselves or by using MPEX `HOOKed` programs).

Not changing your father process is only a problem if the father is still active (VERY unusual). Not changing your sons and "brothers" is a problem IF they remain active or you re-activate one of them to do something. An example of this would be if you were to, within MPEX, run `QEDIT`, suspend it, `%CHLOGON`, re-activate `QEDIT`, and try and edit files.

For this reason, on MPE/iX systems, we do the following:

- If you have any son processes MPEX will not allow you to do a `%CHLOGON`.
- You must first `%KILL` your son processes.
- When you do a `%CHLOGON`, we disable the `[BREAK]` key until you switch you back to your original logon.
- If you exit MPEX (or `STREAMX`) without switching back to your original logon first, we switch you back automatically.

%COPY

Syntax:

```
%COPY fromfileset [,tofileset]
      [;altfileparm] [ ;... ]
      [;{YES|NO|ASK}]
      [;KEEPATTR]
      [;KEEPAMDATES]
      [;KEYFILE= keyfileset]
      [;DEV= [[[remotesystemname]]#] devicename]
      [;COPYACD]
      [;BUFFERSIZE= nnn]
      [;CREATE[Q]]
```

Examples:

```
%COPY @.@.AP, @/=.@.APBACKUP; KEEPATTR; KEEPAMDATES
%COPY APDB@, DBNEW@; DEV=3
%COPY @.DEV, @.SOURCE; ASK
%COPY K@.DATA, N@.DATA; FLIMIT=EOF*1.5; BLKFACT=BEST
%COPY KSAMFILE, NEWDATA.NEWG.NEWA; KEYFILE=NEWKEY.NEWG.NEWA
%COPY APDB@, =; DEV=SYSB#          << to another system >>
%COPY APDB@, =; DEV=#             << back to the host system >>
%COPY ./source/project-3/, ./archive/= ;CREATEQ
%COPY MPEFILE, /posixdir/posixfile ;BYTE
%COPY ./MyDir/MyFile, /YOURACCT/YOURGRP/YourDir
      (Copies "MyFile" to "/YOURACCT/YOURGRP/YourDir/MyFile")
```

The %COPY command allows you to:

- Copy filesets (temporary or permanent);
- Copy IMAGE databases and KSAM files;
- Do all copies (even those of databases and KSAM files) very fast;
- Copy things into other accounts (if you have SM capability);
- Specify new file parameters for the newly-built files (e.g. ;DEV=3 or ;FLIMIT=EOF*1.5 — all of these are the same as those permitted by the %ALTFILE command);
- Copy files (including IMAGE databases!) to other HP3000 systems across dslines.
- Copy files between MPE and POSIX directories, changing the format from FIXED, VARIABLE or UNDEFined to BYTEstream (or the other way around) as needed.
- Create a new POSIX directory when copying files, avoiding the need to manually build a complex directory structure.

This command copies all the files in "*fromfileset*" into correspondingly-named files in "*tofileset*". On systems that support POSIX, if the "*tofileset*" is a subdirectory, then the files are copied into that subdirectory with the same name as the original. If you would like to copy a file to a back-referenced file (prefixed by "*"), you should use the %FCOPY or %PRINT command instead.

What if the "to-file" already exists? (;YES|NO|ASK)

- If the ;YES keyword was specified, the old to-file will be purged.
- If the ;NO keyword is specified, the old to-file will be kept and the copy (of this file) will not be done.
- If the ;ASK keyword is specified, the user will be asked what he wants to do with the old to-file — purge it or not purge it.
- Default is ;ASK online, ;YES in batch.

How is the "to-file" built?

By default, the to-file is built by **FOPENing** it with the same "vital attributes" (device class, FLIMIT, etc.) that the from-file has. This means that the to-file will have the same attributes except for:

- the creator ID (which is set to the copying user's user ID);
- the creation/restore/access/modify dates and times (which are set to the current date and time);
- the file security (which is set to the system default — :SECURED, file-level access mask (R,A,X,W,L:ANY));
- the lockword, which is set to whatever is specified in the "to-fileset".

If you specify a ;KEEPATTR on the %COPY command, the creator ID, the security, and the creation and restore dates won't be changed. All the to-file's attributes except for

- the last modify and access dates and times and
- the lockword

will be the same as the from-file's. If you want to also make the last modify and access dates and times be the same as from-file's, specify BOTH a ;KEEPATTR keyword and a ;KEEPAMDATES keyword. (Note that neither ;KEEPATTR nor ;KEEPAMDATES will work if you're copying to another system, since we can't change the file label of a remote file. However, ;KEEPAMDATES will still preserve the last access date of the from-file.)

You may also use any %ALTFILE keyword to explicitly change any attribute of the target file. For instance, saying

```
%COPY APDB@, DBNEW@; DEV=3
```

will copy all the files whose names start with APDB (presumably the APDB database) into files where the APDB is replaced by DBNEW; but all the to-files will be built on device 3.

Similarly,

```
%COPY K@.DATA, N@.DATA; FLIMIT=EOF*1.5; DELACD
```

will copy K@.DATA into N@.DATA, setting the new files' file limit. Note that these parameters are exactly identical (in syntax and in meaning) to the %ALTFILE command parameters.

This feature, along with the new %ALTFILE keywords of ;FIXED, ;VARIABLE, ;UNDEF, and ;BYTE will be most useful for systems that support POSIX. Most MPE editors and utilities expect FIXED or VARIABLE length records and may corrupt BYTEstream files. POSIX utilities

and programs, on the other hand, will return an error when attempting to read FIXED or VARIABLE files and will only create BYTEstream files as output files. When using the %COPY command to move files between MPE groups and POSIX subdirectories, you will almost always want to add either ;FIXED or ;VARIABLE or ;BYTE to convert the file format as you copy. (Note, however, that this will slow down the speed of the %COPY command).

Special note on private volumes

Because the %COPY command is intended to make an exact copy of the file, preserving virtually all attributes of the file (which is normally what you want it to do), problems may arise (involving the device class specification) when trying to %COPY a file to/from an account residing on a private volume set to/from an account on the system volume set or a different private volume set.

Fortunately, you can easily get around the problem by using the ;DEV= keyword.

For example:

```
%COPY @.PVGROUP.PVACCT, =.MYGROUP.SYSVOL; DEV=DISC
```

Copying into other accounts

The %COPY command lets users with SM capability copy files into other accounts.

For instance, the

```
%COPY @.@.AP, @/=.@.APBACKUP; KEEPATTR
```

command will copy all the files in the AP account into the APBACKUP account (assuming all the appropriate groups are already built in the APBACKUP account). What can we say about this command?

- If the person executing it has SM (perhaps by running GOD.PUB.VESOFT), it'll work even if he's not logged on to the APBACKUP account.
- The target fileset is specified not as "@.@.APBACKUP" but as "@/=.@.APBACKUP". If we'd said "@.@.APBACKUP", this would mean that the to-files would be built without lockwords (much like an ordinary :FCOPY... ;TO=*file.group.acct* will always build the ;TO= file without a lockword, even if the ;FROM= file has a lockword).

Specifying a "/"= after the to-filename ("@") means that all to-files are to be built with the same lockword as their corresponding from-files.

Actually, we could have specified any lockword, e.g. "@/FOO.@.APBACKUP" — this would build all target files with lockword FOO. However, this is usually not as useful as saying "@/=.@.APBACKUP".

- We said ;KEEPATTR, telling MPEX to retain the creator ID, the security information, and the create and restore dates and times. This is often a very good idea for such "bulk" copies.

Note that if we hadn't said ;KEEPATTR, all the to-files would have the creator ID equal to the copying user's user ID. This might cause problems if such a user doesn't exist in the target account.

Copying databases

Databases can be copied with the %COPY command just like ordinary MPE files, e.g.

```
%COPY APDB@, DBNEW@; DEV=3
```

In order to copy a database, you must be logged on as the creator of the database, the account manager of the account containing the database or (if you are copying to or from another account), as the system manager. Internally, MPEX has to do a little bit of work to make sure the root file remains consistent, but you'll never have to see it. Database copies work well and quickly. (Of course, you shouldn't try to %COPY an open database — it won't corrupt the original database, but it is quite likely that the copy will have broken chains or other errors).

Copying KSAM files (;KEYFILE=)

Note: The following discussion relates to Compatibility Mode KSAM files only. Native Mode KSAM files (which no longer have a separate keyfile) are treated by MPEX the same as other files.

The %COPY command can copy KSAM files (very quickly, in fact). For example, if you say

```
%COPY @.DATA, @.NEWDATA
```

and there happen to be some KSAM files in the DATA group, they'll be copied into the NEWDATA group just as normal files would be.

However, remember that a KSAM file is actually not one file, but two (the data file and the key file). How does MPEX figure out what name to assign to the new key file?

In the "%COPY @.DATA, @.NEWDATA" example, the answer is simple — since the to-file file names should be the same as the from-file file names (except for the different group name), MPEX will build the new to-keyfile with the same name as the old from-keyfile.

Similarly, if you say

```
%COPY DEV@.PUB, PROD@.PUB
```

and there's a KSAM file called DEVDATA.PUB with key file DEVKEY.PUB, MPEX will copy it into a file called PRODDATA.PUB with key file PRODKEY.PUB.

However, what if you just say

```
%COPY OLDDATA, NEWDATA.NEWGRP
```

— how can MPEX figure out a new name for the to-keyfile? It can't, which is why in this case it requires that you specify the to-keyfile name explicitly:

```
%COPY OLDDATA, NEWDATA.NEWGRP; KEYFILE=NEWKEY.NEWGRP
```

In general, whenever you copy a single KSAM file (rather than a fileset), you should specify the ;KEYFILE= parameter.

There are also some other cases in which MPEX can't figure out the to-keyfile name. For instance, say that your data file is called DATAAP, your key file is called KEYAP, and you enter the command

```
%COPY DATA@, NDATA@.NEWSTUFF
```

How can MPEX figure out the to-keyfile name? Again, it can't — in fact, MPEX can only figure out the to-keyfile name automatically when the from-keyfile is itself part of the from-fileset OR when the to-fileset contains an "=" (e.g. "%COPY DATA@, =.NEWSTUFF", in which case MPEX will know to keep the same file name but put the file into the NEWSTUFF group).

In the "%COPY DATA@, NDATA@.NEWSTUFF" case, you again have to use the ;KEYFILE= parameter, e.g.

```
%COPY DATA@, NDATA@.NEWSTUFF; KEYFILE=NKEY@.NEWSTUFF
```

Now MPEX will know to copy DATAAP into NDATAAP.NEWSTUFF and build the key file as NKEYAP.NEWSTUFF.

Copying to other computers (;DEV=envid#)

By specifying a DS device or an NS environment ID (followed by a #) as a ;DEV=... parameter, you can copy filesets (including databases!) to other systems over DSLINES or LANs. (You can not, however, copy files from other systems — you must run MPEX on the system you wish to copy from.) For instance, you can say

```
%REMOTE:SYSB HELLO ANNE,MANAGER.PROD
%COPY MYDB@, =; DEV=SYSB#
```

This will copy files in the fileset MYDB@ (presumably a database root file and its datasets) into files with the same name on the system accessed with the SYSB DSLINE.

If you do the :REMOTE HELLO before running MPEX, you will need to enter a %DSLIME command within MPEX before you can do the COPY...; i.e.

```
:REMOTE:SYSB HELLO ANNE,MANAGER.PROD
:RUN MAIN.PUB.VESOFT
%DSLIME SYSB
%COPY MYDB@, =; DEV=SYSB#
```

If you have problems, you might also try specifying the ;BUFFERSIZE= keyword:

```
%COPY MYDB@, =; DEV=SYSB#; BUFFERSIZE=2048
```

to control the number of words we try to copy at a time (the default is 4096 — it's possible that smaller numbers might work better over DS).

Also note that when you copy to a remote machine, the account into which you're copying should be your remote logon account. Non-remote %COPY can copy across account boundaries, but remote %COPY can't (because of MPE restrictions).

Preserving last access/last modify dates (;KEEPAMDATES)

When you %COPY a file, the from file's last access date is set to today's date (because the file is, after all, being accessed); this is the same as MPE/iX's :COPY command does.

However, sometimes you do not want the last access date to be updated – you might want to use it for archival purposes, periodically saying something like

```
%MPEXSTORE @.@.@-@.@.SYS-@.@.TELESUP (ACCDATE<TODAY-180) ;*TAPE;PURGE
```

to clean up your system and save disk space. If you do this, you'd like to update the last access date as rarely as possible — update it only when somebody actually tries to access the file for something productive. If you merely say

```
%COPY @.@.AP,@/=.@.APBACKUP
```

you don't want all the AP account files to be marked as having been accessed today.

If you say

```
%COPY @.@.AP,@/=.@.APBACKUP;KEEPAMDATES
```

then MPEX will do the copy without updating the from-file's last access date.

If you want this to be the standard behavior of the %COPY command, you can say

```
:SETJCW MPEXCOPYKEEPAMDATES=1
```

— this tells MPEX that the %COPY command should never update the last access and last modify dates of files even if ;KEEPAMDATES is not specified. If you're going to use this SETJCW command, you should probably put it into the MPEXMGR.PUB.VESOFT file so that it comes into effect for all users in the system.

If you specify both the ;KEEPATTR and the ;KEEPAMDATES keywords, e.g.

```
%COPY @.@.AP,@/=.@.APBACKUP;KEEPATTR;KEEPAMDATES
```

then the to-file's dates (access, modify, create, and restore) will all be set to be exactly the same as the from-file's dates. A normal ;KEEPATTR without a ;KEEPAMDATES would set the to-file's create and restore dates to be the same as the from-file's, but will make the to-file's access and modify dates be today's date.

If you specify ;KEEPAMDATES without ;KEEPATTR, all of the to-file's dates will be set to today's date (since the to-file is, after all, being created today) — however, the from-file's last access date will still be preserved (on the from-file).

Note that for technical reasons, MPEX will always update the last access and modify dates of a KSAM key file when a KSAM file is %COPYd.

Preserving ACDs (;COPYACD)

Using Access Control Definitions (ACDs), MPE permits you to define exactly who is authorized to access a particular file in a particular way. You can, using the :ALTSEC command, impose an ACD on a file (MPEX's %ALTSEC command lets you do this to multiple files) — this ACD can

indicate that, say, `JOE.PROD` can read and write to this file, while `CLERK.PROD` and `MGR.AP` can only read it, and no other users can access it in any way.

The `;COPYACD` keyword of the `%COPY` command lets you copy a file's ACD together with the file, so that the newly-created file has the same ACD as the original. (The default, just as in MPE's `:FCOPY`, is to build the new file without any ACDs.) Thus, if you want to copy an entire group of files while preserving all their attributes (including lockword, creator ID, access dates, and ACDs), you can say

```
%COPY @.DATA,@/= .NEWDATA;KEEPATTR;KEEPAMDATES;COPYACD
```

If the `%COPY` command encounters a file that has no ACD, it will copy it without an ACD; if, however, it encounters a file whose ACD you are not authorized to read (because you weren't granted RACD access to this file), `%COPY` will refuse to copy the file. This way you'll never be surprised by a file suddenly "losing" its ACD.

The %COPY command's speed

The `%COPY` command is not only much more powerful than `FCOPY` — it's much faster, too. It copies all files (MPE, KSAM, and IMAGE) via MR NOBUF, which is a very efficient copying mechanism.

Copying single files

One other thing that you can do to make `%COPYs` of single files even faster is to prefix "`COPY`" with a `:`, e.g.

```
:%COPY MYFILE,TOFILE;YES
```

Prefixing `%COPY` with a `:` tells MPEX that this is a single-file-only command; MPEX can then save some of its fileset processing overhead. However, this means that the from and to-files must be ordinary MPE filenames — no filesets, selection conditions, `"=`'s in the to-file, etc. Also, the `BLKFACT=...` and `FLIMIT=...` keywords can not be used on a `:%COPY` command.

Lockwords and %COPY

As we discuss in the MPEX And Your System Security section, MPEX is quite intelligent in its lockword handling. If you have SM capability (or AM capability in your own account), MPEX will automatically supply the lockword for each file being processed. This lets you say something like:

```
%COPY @.OLDGROUP,@.NEWGROUP
```

without having to specify the lockword of every lockworded file in `OLDGROUP`.

However, if you execute this `%COPY` command, what happens to all the lockwords of the files being copied?

Well, remember that we must stay compatible with MPE's `:FCOPY` command (as well as with `:RENAME` and with MPE/iX's `:COPY`). If you say (in MPE)

```
:FCOPY FROM=MYFILE/SECRET.OLDGROUP;TO=MYFILE.NEWGROUP;NEW
```

then MYFILE will be copied from OLDGROUP into NEWGROUP BUT THE NEW FILE WON'T HAVE A LOCKWORD. Don't blame us — this is how MPE does it; if the target filename does not include a lockword, the new file will be unlockworded.

Therefore, if you say

```
%COPY @.OLDGROUP, @.NEWGROUP
```

MPEX will interpret this to mean "copy the OLDGROUP files into files in NEWGROUP without lockwords" (just like MPE's :FCOPY).

What if you want to keep the lockwords? No problem! Just say

```
%COPY @.OLDGROUP, @/= .NEWGROUP
```

The "=" in the lockword field means "keep the same lockword as the old file" (see "[Target filesets](#)" in the MPEX Filesets chapter for more information on "=").

Therefore, when MPEX issues the :COPY command for each file in @.OLDGROUP it'll be certain to include the old file's lockword (if any) on the new filename.

Instead of saying the equivalent of

```
:FCOPY FROM=MYFILE/SECRET.OLDGROUP;TO=MYFILE.NEWGROUP;NEW
```

it'll say

```
:FCOPY FROM=MYFILE/SECRET.OLDGROUP;TO=MYFILE/SECRET.NEWGROUP;NEW
```

and all the lockwords will be preserved.

Copying data into existing files

As you see, the %COPY command is really useful for copying files into new files — it won't, say, append a file to an already existing file or fill an existing KSAM file with data from an MPE file. If %COPY finds that the target file already exists, it will either fail (if you specify ;NO) or purge the existing file (if you specify ;YES).

If you want to copy data into existing files, we recommend that you use the %FCOPY command (see [below](#)).

%DBxxxALT commands

With the help and cooperation of several other vendors, we are able to provide MPEX with the ability to change the capacity of IMAGE databases using their products. Note that since these vendors develop their programs independently of VESoft, they may make changes to their programs that MPEX won't recognize. While we do our best to ensure that we are using the current syntax of their commands, we cannot guarantee this will always be the case.

To use any of these commands to change a database's capacity, you must be logged on as the database creator (in the same group as the database), and you must have Read, Write, and Lock access to the database. If you do not meet these conditions, the %DBxxxALT command will try to stream a job for you (via the MPEX %SUBMIT command) that logs on as the database creator and does the capacity change. You will be prompted for the passwords as appropriate.

If your copy of database utility is not in the default group and account where it is normally installed, you should add one of the following SETVAR commands to your MPEXMGR.PUB.VESOFT initialization file to inform our software of the location of your database utility:

```
SETVAR MPEXDBADGFILE "ADAGER.group.account" (for ADAGER)
SETVAR MPEXDBGENFILE "DBGENRL.group.account" (for DBGENERAL)
```

%DBADGALT

Syntax: `%DBADGALT dbfileset, ratio`

Examples: `%DBADGALT @.DB, .50`
`%DBADGALT @.DB (DBSETFULLNESS>.8 and MODDATE>TODAY-30), 0.6`

The %DBADGALT command uses ADAGER to change the capacity of IMAGE datasets so that they are a given percentage full (this command will only work if you own ADAGER, the "Adapter/Manager for IMAGE Databases", a product of Adager).

For instance, the command in the above example increases the capacity of all datasets in the DB group that are currently greater than 80% full so that they are only 60% full.

%DBGENALT

Syntax: `%DBGENALT dbfileset, ratio`

Examples: `%DBGENALT @.DB, .50`
`%DBGENALT @.DB (DBSETFULLNESS>.8 and MODDATE>TODAY-30), 0.6`

The %DBGENALT command uses the DBGENERAL utility to change the capacity of IMAGE datasets so that they are a given percentage full (this command will only work if you own DBGENERAL, which is a product of BRADMARK Technologies, Inc.).

For instance, the command in the above example increases the capacity of all datasets in the DB group that are currently greater than 80% full so that they are only 60% full.

%DEALLOCATE

Syntax: `%DEALLOCATE fileset`

Examples: `%DEALLOCATE @.@.(ISCMProg)`

Deallocates program files. Since only compatibility mode program files can be allocated, use of the file attribute variable ISCMProg as shown here will speed up the execution of this command.

See [%ALLOCATE](#) for more details on allocating programs.

%DELETEALARM

Syntax: `%DELETEALARM alarmnumber`

Examples: `%DELETEALARM`

`%DELETEALARM` deletes the alarms you set via the `%ALARM` command. For more information, see the documentation on [%ALARM](#).

%DELETESPOOLFILE

Syntax: `%DELETESPOOLFILE spoolfileset`

Examples: `%DELETESPOOLFILE $STDLIST.@.@(SPOOL.OUTPUTPRI<=2)`
`%DELETESPOOLFILE @.@.DEV-RPT@.@.(SPOOL.READYDATE<TODAY-2)`
`%DELETESPOOLFILE $STDLIST.@.DEV(SPOOL.JSNAME MATCHES "COM@")`

MPEX's `%DELETESPOOLFILE` command lets you:

- Delete entire filesets of spool files at once;
- Find the spool files to be deleted by their "vital attributes" (spool file name, creating job name, output priority, creation date, etc.);
- Delete spool files if you
 - are console operator or
 - have SM or OP capability or
 - have AM capability and are deleting spool files in your own account or
 - are the spool file's creator.

The command

```
%DELETESPOOLFILE $STDLIST.@.@(SPOOL.OUTPUTPRI<=2)
```

will delete all job \$STDLISTs whose output priority is 2 or less;

```
%DELETESPOOLFILE @.@.DEV-RPT@.@.(SPOOL.READYDATE<TODAY-2)
```

will delete all DEV account spool files made ready more than 2 days ago except for those whose spool file names begin with RPT;

```
%DELETESPOOLFILE $STDLIST.@.DEV(SPOOL.JSNAME MATCHES "COMP@")
```

will delete all DEV account \$STDLISTs created by jobs whose job names start with COMP (presumably the listings of compile job streams).

MPE/iX native mode spooler note:

The Native Mode Spooler on MPE/iX does not retain the spoolfile's group name. If you specify a group name in %DELETESPOOLFILE on MPE/iX, it will be effectively ignored. For example:

```
%DELETESPOOLFILE $STDLIST.PUB.PROD
```

is the same as

```
%DELETESPOOLFILE $STDLIST.@.PROD
```

%DELETETELVAR

Syntax: `%DELETETELVAR varset [,...]`

Examples:

```
%DELETETELVAR COUNTER
%DELETETELVAR J, K, L
%DELETETELVAR K#, FOO@
```

The %DELETETELVAR command deletes local variables. Please see "[Local variables](#)" in the MPEX Variables chapter of this manual for more details.

%DELETEVAR

Syntax: `%DELETEVAR varset [,...]`

Examples:

```
%DELETEVAR X1
%DELETEVAR TEMPV@, MYFILENAME
```

%DELETEVAR deletes all the specified MPEX variables. You may use wildcards (@, # or ?) to delete all the variables whose names match a given pattern. For more information on this command, see the [MPEX variables](#) chapter.

%DEMO

Syntax: `%DEMO [section]`

Examples:

```
%DEMO
%DEMO 3
```

This command starts the MPEX 'self-demo', which shows you the major features of MPEX. The demo consists of an introduction and 5 sections (covering each of the major areas where MPEX enhances your system). You may stop the demo at any time, and restart it later at the beginning of any section.

%DEVCONTROL

Syntax: `%DEVCONTROL ldevnumber; {LOAD|ONLINE}`

Examples: `%DEVCONTROL 7;ONLINE`

One of the few drawbacks of HP's DDS tape drives is that they have no "online" button. In order to put a tape back online, you must first eject the tape and then re-insert it.

The %DEVCONTROL command uses the HPDEVCONTROL intrinsic to either load a mounted tape or put a loaded tape online.

Many new backup systems and devices have the ability to do a full system backup without an operator present to mount additional tapes. However, once the backup is over, you must wait for an operator to put the tape back "online" in order to validate that all files were successfully stored. After your backup is done, you can use %DEVCONTROL to put the tape back online, then verify your backup (using MPE/iX's :VSTORE command), all without an operator present!

%DO

Syntax: `%DO [abscmdnum] [,editstring]
 [-relcmdnum]
 [cmdnum1/cmdnum2]
 [["][@]string["]]`

Examples: `%DO 155
 %DO -2
 %DO 155/158
 %DO RUN
 %DO @MYFILE`

%DO re-executes a command that you typed earlier and that has been saved in the MPEX command history.

%DO, %LISTREDO, and %REDO are related, and are discussed in full detail under the %REDO command in this manual.

%DOSAVED

Syntax: `%DOSAVED ;FILE= savefile
 [;NOVERIFY]
 [;QUIET]
 [;MUSTEXEC]
 [;mpeschedparms]`

Examples:

```
%DOSAVED;FILE=SAVEDJBS
%DOSAVED;FILE=PRODJBS; NOVERIFY; AT=23:30
%DOSAVED;FILE=ALLJOBS; QUIET; MUSTEXEC
%DOSAVED;FILE=CRITJOBS; IN=0,0,5
```

%DOSAVED re-submits jobs that were saved via the %SAVEJOB command. For more information, see the documentation on [%SAVEJOB](#).

%ECHO

Syntax:

```
%ECHO [message]
```

Examples:

```
%ECHO PLEASE WAIT WHILE WE RUN AP000
%ECHO
%ECHO !PROCESSEDFILES files processed, !FAILEDFILES errors
```

The %ECHO command outputs the specified message to the terminal. It's most useful when you're executing a UDC or a command file and want to tell the user something.

%ECHO is also useful for outputting the values of variables, since "!"s and "![...]"s are processed before the message is output. See the chapter on [MPEX variables](#) for more information.

If you want to output text with leading spaces, just insert the right number of spaces (plus the one that always goes after the %ECHO) between the %ECHO and the message. %ECHO is the only command that actually cares how many spaces there are between it and its parameters.

%EDIT

Syntax:

```
%EDIT fileset, editcommands
```

Examples:

```
%EDIT @.SOURCE(CODE="EDTCT"), SET VARIABLE
```

The %EDIT command lets you perform an EDITOR/3000 command (or multiple commands separated by semicolons) on an entire fileset:

- To change all occurrences of one string to another in a fileset, say

```
%EDIT @.SOURCE, CHANGE "ACCT-TYPE", "ACCOUNT-TYPE", ALL
```

MPEX will tell EDITOR to /TEXT each file, do the /CHANGE, and then /KEEP it back (note that the /KEEP will happen regardless of whether or not the /CHANGE actually changed anything — see the [%EDITCHG](#) command for a better solution).

- To convert your fixed record-length COBOL source files to variable record-length files (and save up to 30-50% of their disk space!), say

```
%EDIT @.SOURCE(CODE="EDTCT"), SET VARIABLE
```

Note that this works only for COBOL files or UNNUMBERED files; NUMBERED non-COBOL files will be rendered uncompileable by an EDITOR /SET VARIABLE command — if you accidentally do this to them, just do another %EDIT command to do a /SET FIXED.

When will %EDIT "/KEEP" the file?

Naturally, we didn't implement all the functions of EDITOR inside MPEX. For each file in the fileset, MPEX actually runs EDITOR and passes to it:

- A /TEXT command to text in that file;
- The command you specified (/LIST, /WHILE, /CHANGE, etc.);
- Possibly a /KEEP command to keep the file back;
- An /EXIT command to return control back to MPEX.

When does MPEX tell EDITOR to /KEEP a file? Well, if your EDITOR command includes only those commands that MPEX knows will never change the file (e.g. /LIST, /WHILE, /FIND, etc.), MPEX will not tell EDITOR to /KEEP the file.

If, however, there's any chance that the file may be changed by your command — for instance, if your command is a /CHANGE, a /JOIN, a /SET, etc. — MPEX will tell EDITOR to do the /KEEP, to prevent your changes from getting lost.

If you know that you don't want to do a /KEEP, you can just append ";EXIT" to your command, e.g.

```
%EDIT @.SOURCE, CHANGE "ITEM-NUM", "ITEM-NUM", ALL; EXIT
```

This will cause EDITOR to /EXIT before it gets a chance to do the /KEEP.

Actually, there's little harm in doing an unnecessary /KEEP; file parameters (creator ID, creation date, modify date, etc.) will get changed, but the data will remain intact. Still, it's a good idea to avoid doing /KEEPS unless you need to (if only for speed's sake) — this is why MPEX automatically avoids /KEEPS in certain cases and lets you explicitly avoid them yourself by specifying a ";EXIT" at the end of your EDITOR command.

%EDITCHG

Syntax:

```
%EDITCHG fileset, "fromstring", "tostring" [,quotechar]
```

Examples:

```
%EDITCHG @.SOURCE, "ACCT-TYPE", "ACCOUNT-TYPE"
%EDITCHG @.SOURCE, "DD'MM'YY", "DD'MM'YY", \
```

Saying

```
%EDITCHG @.SOURCE, "ACCT-TYPE", "ACCOUNT-TYPE"
```

does exactly the same thing as saying

```
%EDIT @.SOURCE, CHANGE "ACCT-TYPE", "ACCOUNT-TYPE", ALL
```

— it changes all occurrences of "ACCT-TYPE" to "ACCOUNT-TYPE" in all SOURCE group files.

However,

- It will only /KEEP those files that are actually changed.

Any files that do not contain the string "ACCT-TYPE" are not /KEEPed back — their creation and last modify dates aren't changed, and neither are any other parameters (like file code, record type, etc.) that EDITOR is prone to changing.

Thus, for doing EDITOR /CHANGES, the %EDITCHG command is both more efficient and more convenient — for instance, since the last modify date of the files that don't have "ACCT-TYPE" is not changed, those files won't be automatically included on daily stores.

What if the string you want to change contains both single- and double-quote characters? EDITOR allows you to use some character that doesn't occur anywhere in the string as a string delimiter (e.g. a "\"), but MPEX doesn't. To tell %EDITCHG to use a different quote character, include it as the third parameter; the "*fromstring*" and "*tostring*" must still be enclosed in matching quotes (either single ' or double "). (To embed quotes in a quoted string within MPEX, you simply type the quote character twice.)

For example, to change the string

```
CODE 'VAL:="A";
```

to:

```
CODE 'VAL:="B";
```

in the fileset "@.SOURCE", use the command

```
%EDITCHG @.SOURCE, 'CODE 'VAL:="A"', 'CODE 'VAL:="B"', \
```

(Notice that you do not use the "\"" character in the "*fromstring*" or "*tostring*" yourself; %EDITCHG does it for you.)

%EDITQUAD

Syntax: `%EDITQUAD fileset, quadcommands`

Examples: `%EDITQUAD @.SOURCE, C "ACCT-TYPE", "ACCOUNT-TYPE"`

(QUAD is a popular contributed text editor.) If you have QUAD set up as QUAD.PUB.VESOF, MPEX's %EDITQUAD command will execute a QUAD command (or multiple commands separated by semicolons) on a fileset.

MPEX will, for each file in the fileset, run QUAD.PUB.VESOF, tell it to /TEXT in the file, execute the commands you gave, do a /KEEP, and do an /EXIT.

If you don't want QUAD to /KEEP the file, append an ";EXIT" to *quadcommands*, e.g.

```
%EDITQUAD @.SOURCE, LIST ALL OFFLINE; EXIT
```

Otherwise, MPEX will always tell QUAD to /KEEP the file.

The "<" and ">" characters are special characters in MPEX (and MPE/iX) indicating I/O re-direction (see [Redirecting command input and output](#) in the MPEX User Manual). QUAD uses these characters to indicate a "Range File". In order to use them in an %EDITQUAD command, you must prefix them with two "!" characters (not just one). For example:

```
%EDITQUAD WORK@, COPY !!<NEWSTUFF!!> TO 4
```

%EDITTDP

Syntax: `%EDITTDP fileset, tdpcommands`

Examples: `%EDITTDP @.SOURCE, FIND "ITEM-NUM"; EXIT`
`%EDITTDP @.SOURCE, CHANGE "TO", "TWO", ALL`

Just as the %EDIT command lets you execute an EDITOR command on a fileset, so the %EDITTDP command lets you execute a TDP command (or several commands separated by semicolons) on a fileset.

For each file in the fileset, MPEX tells TDP to:

- /TEXT it,
- Execute the specified command (or commands),
- /KEEP it, and
- /EXIT.

If you want to just execute the command and not do a /KEEP (for instance, if the command is a /LIST ALL,OFFLINE), you should append an ";EXIT" to your TDP command:

```
%EDITTDP fileset, LIST ALL,OFFLINE; EXIT
```

This will make TDP do the /EXIT before it gets a chance to do the /KEEP. If you don't specify the /EXIT, the /KEEP will always be executed (even for /LIST commands).

%ERASE

Syntax: `%ERASE fileset`

Examples: `%ERASE @.DATA`

The %ERASE command erases the contents of the files while leaving the file structure intact (file size, ASCII/BINARY, record size, etc.); when you are done, the files will continue to exist (they simply won't contain any data).

%ERRCLEAR

Syntax: `%ERRCLEAR`

Sets the variables `CIERROR`, `CIERRORISWARN`, `FSERROR`, `HPCIERR` and `HPFSERR` to 0.

%ESCAPE

Syntax: `%ESCAPE [[CIERR=] errnum]`

Examples: `%ESCAPE`
`%ESCAPE 985`

Terminates all levels of the currently executing UDC/command file/batch job (unless there is a `%CONTINUE` or `%TRAPERROR` in effect), sets `CIERROR` to the absolute value of `errnum`, and sets `HPCIERR` to the value of `errnum`.

- If there is a `:CONTINUE` in effect, execution continues with the next command.
- If there is a `%TRAPERROR` in effect, execution continues from the next `%IFERROR` or `%CLEANUP`.

%EXIT

Syntax: `%EXIT`

`%EXIT` will terminate the active MPEX process, unless it's being run from within another program, in which case it will suspend MPEX instead of terminating it. MPEX will not attempt to suspend when being run from within the command interpreter.

If you need to terminate MPEX without attempting to suspend, use the `%QUIT` command. If you need to terminate both MPEX and the current session, use the `%BYE` command.

Two synonyms for `%EXIT` are also supported: both `%E` and `%END` have the same effect as `%EXIT`.

%FCOPY

Syntax: `%FCOPY [FROM=] fromfileset, [TO=] tofileset
[; fcopyparms]`

Examples: `%FCOPY S@.SRC+P@.PUB, =.OLD; NEW`
`%FCOPY FROM=RPT@.DATA; TO=SEL@.=; NEW; SUBSET="X7"`

The `%FCOPY` command lets you copy filesets using `FCOPY`. This is useful if you want to use `FCOPY`'s special features, like `;SUBSET=`, `;CHAR=`, etc.

MPEX COMMAND REFERENCE: %FILTER

Simply put, MPEX's %FCOPY runs FCOPY for each file in the fileset and tells it to do a copy ;FROM= *fromfile*;TO= *tofile* with the parameters you specified.

If the first parameter in the "*fcopyparms*" is "NEW", MPEX will automatically purge the target file. Other than that, typing this command is identical to saying:

```
%REPEAT
%>:FCOPY FROM=!MPEXCURRENTFILE;TO=[OBJECTFILE('tofileset')];fcopyparms
%>FORFILES fromfileset
```

(see the [%REPEAT...%FORFILES](#) section for details on this command and the OBJECTFILE(...) function).

%FILTER

Syntax:

```
%FILTER "command" [,searchexpression [;mpexprintparms]]
```

Examples:

```
%FILTER "SHOWJOB", " 113 "
%FILTER "RUN MYPROG", CL"TOTAL"
%FILTER "%COBOLII @.SOURCE,=.PUB", "ERROR"; CONTEXT=-2
%FILTER "SHOWDEV", "DOWN PENDING" OR "DP"
```

Have you ever done a SHOWJOB in order to find out who was logged on to a particular LDEV, only to have to wade through screen after screen of data searching for the LDEV number you were interested in? Or run a program that produces a whole bunch of output, when all you're really interested in is the "TOTAL" line?

The "%FILTER" command lets you execute any command that you could normally type at MPEX's "%" prompt, and "filter" the output using the MPEX %PRINT command's powerful ";SEARCH=" feature. (In fact, %FILTER is merely a convenient combination of %PRINT, documented in this manual, and the STDLIST(...) function, documented in Appendix; it has been implemented to make using these features more convenient — take a look at FILTER.CMD22.VESOFT!)

As you can see from the examples above, the first parameter is the command you wish to execute. If it needs to contain spaces, commas, or semi-colons, you must put quotes around it.

The second parameter is the same kind of thing you can pass to the FSEARCHEXP(..) function: a %PRINT...;SEARCH=... type of expression optionally followed by any additional %PRINT parameters (like ;CONTEXT=, ;PREV=, etc.).

For example, to see all of the jobs on your system that are SCHED and have HIPRI (which SHOWJOB displays as inpri 15), you could say

```
%FILTER "SHOWJOB SCHED", " 15 "
```

Note that since %FILTER internally executes the %PRINT command, it is affected by any variables you may have set that would affect %PRINT (like MPEXPRINTNONDELIMS). For more details on the search expression, see "[%PRINT...;SEARCH=...](#)" in the "MPEX Commands" section of this manual.

%GOON, %SHOWGOON

Syntax:

```
%GOON command
%SHOWGOON
```

Examples:

```
%GOON PRINT @.SOURCE;SEARCH="SOC-SEC-RATE";NUM
%GOON ALTFILE @.@.DEV; XLTRIM; KEEPAMDATES
%GOON COBOL MYPROG.SOURCE,=.PUB
```

%GOON (which stands for "go on") lets you execute any command that you could normally type at the MPEX prompt as a son process, immediately returning control of your terminal to you, and re-directing output of the command to a temporary file! This lets you run tasks that may take a long time to complete in the "background", without tying up your terminal. It is similar to the **;GOON** keyword of MPEX's enhanced **%RUN** command (documented later in this manual).

Here is an example of using **%GOON**:

```
%GOON LISTF @.SOURCE.DEV(FSEARCHSTRING("PART-NUMBER")>0),2
Output will be sent to temporary file VE0139
```

After typing **%GOON LISTF...** and pressing [RETURN], you immediately get an MPEX prompt again. This allows you to do other things while this LISTF is executed in the background. The output of **%GOON LISTF...** is redirected to a temporary file named **VE0139** (in this example), so that it doesn't interfere with your display while you are doing other things.

Sometime later, a message will appear (over your function key labels if you have an HP terminal) that says:

```
Execution of LISTF done; see file VE0139
```

to see the results, simply **%PRINT** the output file:

```
%PRINT VE0139
```

As you can see, when you enter an MPEX command using **%GOON**, you will be shown the name of the temporary file that the output will be sent to. When the command is done, a message letting you know it is done will be sent to your terminal (written over your function key labels, if you have an HP terminal, the same way the **%WARNF** command does), telling you which command finished (in case you have more than one **%GOON** running!) and reminding you of the name of the output file for that command.

If you were to **%EXIT** from MPEX before all **%GOON** processes completed, they would be automatically killed. To prevent this, MPEX will not let you exit; if you try you'll see:

```
Error: Son processes still running, you must %KILL them before exiting.
```

Simply use the **%SHOWGOON** command (described below) to see which ones are still active, and wait for them to finish or **%KILL** them.

MPEX COMMAND REFERENCE: %HELP

What advantage does the %GOON command have over MPEX's *\$command*, %SUBMIT... , and %SCHEDULE... , all of which create a job stream to execute the command for you?

- Job streams frequently have to sit in the WAIT queue for a while before actually logging on;
- after WAITing, the job has to log on, which is a high-overhead operation;
- many users launching background tasks may clutter up your job queue and create a lot of \$STDLIST spoolfiles;
- finally, since the %GOON command executes within your session, it has access to all of your MPEX (or MPE/iX) variables and TEMP files.

Checking status of %GOON processes (%SHOWGOON)

%SHOWGOON displays the status of all processes started using %GOON:

```
%SHOWGOON
FILENAME  [STATUS]  COMMAND

VE053     [done]    PRINT @.SOURCE;SEARCH="SOC-SEC-RATE";NUM
VE073     [active]  ALTFILE @.@.DEV; XLTRIM; KEEPAMDATES
VE094     [active]  COBOL MYPROG.SOURCE,=.PUB
```

In this example the %ALTFILE and %COBOL commands are still running, but %PRINT is done. %SHOWGOON does NOT report on processes launched via the %RUN command's ;GOON keyword.

Setting %GOON process priority

By default, %GOON commands are executed in the DS queue, in order to minimize their impact on your system. On most systems, this gives them the same priority as normal batch jobs.

You can change which queue your %GOON processes execute in by setting the variable VESOFTGOONPRI to "CS", "DS" or "ES"; e.g. to have all future %GOON processes that you create (in the current session) execute in the CS queue, simply type:

```
%SETVAR VESOFTGOONPRI "CS"
```

The maximum value that you can actually run %GOON processes at may be limited by the system manager via the %SET GOONMAXPRI command (documented later in this manual).

%HELP

Syntax:

```
%HELP [keyword]           help on MPEX
%SEC HELP [keyword]       help on SECURITY
%VEAUDIT HELP [keyword]   help on VEAUDIT
%HELP :[keyword]          help on MPE
%HELP CIERR number        Displays CI error messages
%HELP FSERR number        Displays FILE SYSTEM messages
%HELP LOADERR number      Displays LOADER error messages
%HELP POSIX
```

Examples:

```
%HELP INTRO
%SEC HELP INTRO
%VEAUDIT HELP INTRO
gets you a general introductory discussion of MPEX, SECURITY or VEAUDIT.

%HELP :SHOWCATALOG
opens the MPE help file, CICAT.PUB.SYS, and displays the normal MPE help on
this command or keyword. Since this uses MPEX's help facility, it means that
MPE's help can be back-scrolled!
```

See also: [%SYNTAX](#), [%SEC SYNTAX](#), [%VEAUDIT SYNTAX](#).

The %HELP command lets you read this manual (and its index!) online.

MPEX's HELP facility is conceptually oriented. What this means to you is that the %HELP command allows you to ask for help on general concepts, rather than on a specific command. There are very many — literally thousands — of keywords you can get help on.

As an example, someone familiar with DOS on a PC would know that the DEL command is used to DELETE files from the disk. Typing DEL in MPE or MPEX, however, returns "UNKNOWN COMMAND NAME". In MPEX, you could type:

```
%HELP DEL
```

MPEX will respond with something like:

```
1. %PURGE: Deletes a file or fileset
2. VEMODIFY: Control-D (Delete character)
3. Spool files: deleting by output priority
4. %DELETEDVAR, MPEX command
5. %DELETEDALARM, MPEX command
6. %DELETESPOOLFILE, MPEX command
7. MPE/V and MPE/iX differences: NM spooler %DELETESPOOLFILE
8. Link to MPE's :DELETESPOOLFILE
9. %DELETEDVAR, MPEX command
10. Link to MPE's :DELETEDVAR
Which of the above would you like help on?
```

You realize that the thing you want is probably item #1 ("%PURGE: Deletes a file or fileset"), so you enter "1". MPEX will now display for you the manual page that contains the description of the %PURGE command. MPEX will then prompt you:

```
(F)orward,(B)ack,(>)forward 1/2 page,<)back 1/2 page,(M)enu,(E)xit?
```

This lets you page forward and backward through the manual to read the entire %PURGE section of the manual (which could be several screens).

Note that when MPEX displays the manual page describing the thing you asked for, the actual description will start about a third of the way down the screen; thus, it displays about eight lines of manual text preceeding the description as well as the description itself. This is done because the text that immediately precedes the description is often (though not always) related information that might be useful. As long as you remember that the thing you asked for actually starts at about line eight or nine of the screen, you shouldn't get confused.

MPEX COMMAND REFERENCE: %HELP

When you're done reading, type "E" to return to the % prompt or "M" to get back to the DELETE keyword menu we showed above (in case you realize you chose the wrong option).

This online HELP index is even more useful than the index in the back of this manual, since the %HELP command finds all the index entries that contain the given keyword (not just those that start with that keyword).

You don't need to specify the full keyword; you might, for instance, say

```
%HELP CRE
```

which will get you a menu like:

1. Spool files: selecting by creating job/session name
2. CREDATE, file attribute variable
3. CETIME, file attribute variable
4. CREATOR, %ALTFILE command keyword
5. CREATOR, file attribute variable
6. Security: removing creator-only restrictions

As you see, index entries containing the strings "creating", "CREDATE", and "CETIME" were found.

In addition to the MPEX commands, the HELP facility will also access the MPE help catalog keywords and topics (see the [%HELPMAKE](#) command for details). These are noted in the menu as "Link to MPE's :COMMAND" where COMMAND refers to an MPE command that matches the keyword that you entered. In the first example, for %HELP DELETE, the MPE commands :DELETESPOOLFILE and :DELETEVAR were listed as well. You can select either of these by number, just as any other MPEX command, and move forward and backward through MPE's help text on these commands.

Getting help on SECURITY and VEAUDIT topics

When you request help on a topic, MPEX normally searches the MPEX manual first. If the topic you are interested in is in the MPEX manual, then the %HELP command displays the topic and does not search any farther. If the topic you requested is not in the MPEX manual, then the SECURITY manual is searched, and if not there, then the VEAUDIT manual, and finally, the MPE or UDC help. Normally, this is exactly what you want; however, there are a number of SECURITY and VEAUDIT topics that are mentioned in this manual, so it is possible that requesting help for what is primarily a SECURITY topic would result in a listing of only those topics explained in the MPEX manual.

In order to force the %HELP system to search the other manuals, preface the command with %SEC (for SECURITY topics) or %VEAUDIT (for auditing related topics).

Getting help on MPE CI, File, and Loader errors

On MPE/V systems, HP supplied a utility called EXPLAIN that explained what a particular CIERROR number means. On MPE/iX systems, the text of each CIERROR is included as part of the CICAT.PUB.SYS file. Our %HELP command makes use of this information to display the "help text" for any given CIERROR.

When you type

```
%HELP CIERR 976
```

MPEX displays the text associated with the error, including suggestions for recovering from the error and avoiding it in the future. Similarly, help on LOADER errors can be found with:

```
%HELP LOADERR 36
```

and FILE SYSTEM (FS) errors with:

```
%HELP FSERR 100
```

You can also get help on the following special keywords:

INTRODUCTION	an introductory discussion of MPEX in general
CONTENTS	online access to the Table Of Contents
FILESETS	information on MPEX filesets
NEWS	new features of MPEX
:SUMMARY	summarizes MPE HELP system commands/topics

Finally, if you just say

```
%HELP
```

MPEX will display this very chapter.

Note that MPEX's %HELP is used primarily for getting help on MPEX, SECURITY, and VEAUDIT features from these manuals as well as displaying the MPE help catalog (CICAT.PUB.SYS) with the ability to scroll forward or backward at will.

If you want help on UDCs or command files (which are not contained in CICAT.PUB.SYS) then you must preface the HELP command with a ":" as follows:

```
:%HELP MPEX
```

%HELPMAKE

Syntax:

```
%HELPMAKE [;NOMPE]
```

Examples:

```
%HELPMAKE

Building MPEX HELP file
Linked in 1190 MPE help keywords
Linked in 22 VEAUDIT help keywords
Linked in 33 SECURITY help keywords
Linked in 196 POSIX help keywords

Building SECURITY HELP file

Building VEAUDIT HELP file
```

The %HELPMAKE command rebuilds and re-indexes our internal help files. By default, the current MPE help catalog (CICAT.PUB.SYS) is indexed as well. This is done automatically when installing or updating our software and when we detect that you have updated the operating system. Although at MPE update time HP's installation job does not know that you have our software, (and therefore does not execute this command), our program can detect that the MPE help catalog has been updated later than our help catalog and therefore needs to be indexed. We only do this the first time you issue a %HELP command in MPEX and the files are 'out of sync'.

%HOOK

Syntax:

```
%HOOK filename
      [;MPEXPREFIX= char]
```

Examples:

```
%HOOK DBUTIL.PUB.SYS
```

This command allows you to "HOOK" programs, so that you can execute MPEX commands from within them! Please see the [MPEX HOOK](#) chapter elsewhere in this manual for more details.

In the rare cases where the "%" character is needed to begin a line in your application, you can specify an alternate "trigger" character to invoke MPEX. Simply add

```
;MPEXPREFIX="x"
```

to the %HOOK command. "x" can be any special character and will default to "%". When you begin a line of input in your application with this character, the HOOK procedure will intercept the line and execute it as an MPEX command.

%IDENTIFY

Syntax:

```
%IDENTIFY filereference, "description"
          [;NOCHECK]
          [;SYS]

%IDENTIFY filereference ;DELETE
          [;NOCHECK]
          [;SYS]
```

Examples:

```
%IDENTIFY MYSOURCE, "My latest program (source code)"
%IDENTIFY MYSOURCE ;DELETE
%IDENTIFY EDITOR.PUB.SYS, "System editor";SYS
```

This command allows you to apply an extended description to a file. We store the extended description in a file called ID.DATA.VESOFT. The extended description will be shown when you use %LISTF *files*set, ID (see the [%LISTF section](#) for details). Although each description may be up to 80 characters in length, you may find it convenient to limit your descriptions to 31 characters to avoid line wrapping when using %LISTF..., ID.

The file specified by filename must exist on the system to be identified or deleted; however, the ;NOCHECK keyword will allow you to override this requirement. Use the ;DELETE option to remove an extended description from the file. Note that this does not :PURGE the file, it merely removes the extended description from our data file.

For your convenience, we have pre-loaded the data file with descriptions of all files found in the VESOFT and VECSL accounts along with the files found in the PUB and HPBIN groups of the SYS account. If you wish to change these descriptions or have additional files in these groups that we have not identified, then you must use the ;SYS keyword.

Security considerations

In order to set or change an identification string for a file, you MUST be either the file's creator, the account manager of the account where the file resides or a system manager. You must have SM capability to use the ;SYS keyword for predefined files.

%IF...%ELSEIF...%ELSE...%ENDIF

Syntax:

```
%IF logicalexpression [THEN]
%   [command]
%   [...]
%[ ELSEIF logicalexpression [THEN]
%   [command]
%   [...]
%[ ELSE
%   [command]
%   [...]]]
%ENDIF
```

Examples:

```
%COMMENT --- Check room in dataset 1
%IF VEFINFO("DBASE01").DBSETFULLNESS > .90 THEN
%   ECHO ERROR: The dataset is more than 90% full!!!
%   ECHO      Aborting...
%   RETURN
%ELSEIF VEFINFO("DBASE01").DBSETFULLNESS > .75 THEN
%   ECHO Warning: the dataset is more than 75% full.
%   INPUT ANSWER; PROMPT="OK to continue? "
%   IF UPS(ANSWER[0:1])<>"Y" THEN
%       RETURN
%   ENDIF
%ELSE
%   ECHO No problem!  You have plenty of room...
%ENDIF
%COMMENT --- Continuing with the update
```

MPEX's %IF and %ELSEIF commands are just like MPE's :IF and MPE/iX's :ELSEIF commands, but with one major enhancement: they give you the full power of VESOFT expressions! (VESOFT expressions are constructed from the hundreds of integer, boolean, string, real, date, and time variables, operators, and functions documented in the Appendix.)

MPEX COMMAND REFERENCE: %INITUDCS

The number of things you can do with VESOFT expressions is practically unlimited; here are just a (very) few examples to get you started:

```
%IF NOT FEXISTS("DATAFILE") THEN
%   ECHO The DATAFILE is missing!
...
```

```
%IF DATEDAY(TODAY+1)=1 THEN
%   ECHO Today is the last day of the month.
...
```

```
%IF VEFINFO("DBROOT").OPENED THEN
%   ECHO The database root file is opened; someone is
%   ECHO currently accessing the database.
...
```

```
%IF UPS(PRINTOPREPLY("OK to start Payroll?")[0:1])="Y" THEN
%   ECHO The console operator has confirmed;
%   ECHO start the Payroll run now.
...
```

%INITUDCS

Syntax:

```
%INITUDCS [NOBUF]
```

Examples:

```
%INITUDCS
```

The %INITUDCS command recognizes what UDCs you have :SETCATALOGed and makes them accessible from within MPEX.

This is usually automatically done whenever you enter MPEX (because the MPEXMGR.PUB.VESOFT file, which is always executed when you enter MPEX, by default contains a %INITUDCS command). However, if you press [BREAK] and do a :SETCATALOG to change your effective UDCs, MPEX won't know about this change.

Entering

```
%INITUDCS
```

will "refresh" MPEX's knowledge of which UDCs are in effect. Thus, a typical use would be:

```
% [you press the [BREAK] key]
:SETCATALOG NEWUDC
:RESUME
%INITUDCS
```

If you say

```
%INITUDCS NOBUF
```

MPEX will read the UDC files MR NOBUF, which could be quite a bit faster than a normal "%INITUDCS".

However, whenever MPEX needs to execute a UDC, MPEX will have to use more stack space, possibly causing STACK OVERFLOW aborts. If you really want to decrease the time it takes to get into MPEX, you might want to replace the %INITUDCS in MPEXMGR.PUB.VESOFT by a "%INITUDCS NOBUF" (see "[Specifying commands to be executed every time MPEX is entered](#)" in the Running MPEX section). Then, if you start getting STACK OVERFLOWS, you'll have to change back to "%INITUDCS".

%INPUT

Syntax:

```
%INPUT [NAME=] varname
      [;PROMPT= "prompt"]
      [;WAIT= waitseconds]
```

Examples:

```
%INPUT MYFILE;PROMPT="Which file do you want to process? "
%INPUT DUMMYVAR;PROMPT="Hit return to continue...";WAIT=3
```

As we discuss in the MPEX Variables chapter, MPEX variables can contain arbitrary data that can be set with the %INPUT and %SETVAR commands and later substituted into MPEX commands.

The %INPUT command prompts the user for input and stores it (as a string) in the variable with the specified name. For instance,

```
%INPUT MYFILE;PROMPT="Which file do you want to process? "
```

will ask the user "Which file do you want to process? " and put the input into the variable MYFILE. (If the ;PROMPT= string hadn't been specified, the user would be asked for input, but wouldn't be shown a prompt string.)

If the variable doesn't exist, it's created; if it exists, its old value is thrown away and it's set to the user-input value.

If the user presses [RETURN], the old value of the variable is unchanged (although if the variable didn't already exist, it is created and set to "").

If you specify ;WAIT=, MPEX will wait for at most "*waitseconds*" seconds for user input; if the user doesn't respond by that time, the read will terminate and the CIERROR JCW will be set to 9003. In the example

```
%INPUT DUMMYVAR;PROMPT="Hit [RETURN] to continue...";WAIT=3
```

we're not so much prompting for data as just waiting for the user to press the [RETURN] key — in this case, we should probably say

```
%NOMSG INPUT DUMMYVAR;PROMPT="Hit [RETURN] to continue...";WAIT=3
```

MPEX COMMAND REFERENCE: %KILL

The %NOMSG prefix tells MPEX not to print an error message if an error occurs — this will prevent the error message "THE INPUT TIMED READ HAS EXPIRED. (CIWARN 9003)" from being printed.

%KILL

Syntax: `%KILL {sonprocessname|sonprocesspin}`

Examples: `%KILL STREAMX.PUB.VESOFT`
`%KILL 54`

MPEX is a powerful process-handling environment (see "[MPEX process handling](#)" in the MPEX Special Topics section of this manual) in which you can manage multiple son processes, active or suspended. For instance, if you say

```
%SPOOK5
```

and then >EXIT out of it, SPOOK5 will remain suspended as a son process of MPEX. Then, when you type

```
%SPOOK5
```

again (assuming you haven't exited MPEX in between the two %RUNs), SPOOK5 will be re-activated rather than re-run from scratch — a much faster and more resource-efficient operation. (Not all programs behave this way; STREAMX does, MPEX itself does, and so do some others.)

Sometimes, you may find that you need to terminate a son process that's suspended (or active — see discussion of the [%RUN;GOON](#) command).

For instance, if you get into SPOOK5, >TEXT in a spool file, and >EXIT back to MPEX, the spool file will stay locked until you re-enter SPOOK5 and unlock it (by doing another >TEXT) or until the SPOOK5 son process terminates (perhaps by doing a >QUIT). At this point, you may say

```
%KILL SPOOK5.PUB.SYS
```

MPEX will look at all its son processes (only in your own session, of course, and only MPEX's immediate sons) and kill the one that's running SPOOK5.PUB.SYS. Alternatively, you might do a

```
%SHOWTREE
```

to find out SPOOK5's PIN (Process Identification Number) and then kill it by specifying the PIN (assuming that 54 is the appropriate PIN):

```
%KILL 54
```

One more reason why you might want to %KILL a son process is if you want to re-run that program with a different run-time parameter (e.g. a different ;PARM=, ;INFO= or entry point).

In this case, you certainly don't want the process to be re-activated; you want to %KILL it and then do the %RUN with the new parameters, which will create a new son process with the parameters specified.

%LISTF

Syntax:

```
%LISTF filesset [:{SPOOL|TEMP}] ,mode [;listfile]
                        [;{NOTOTALS|NORMTOTALS|ALLTOTALS}]
                        [;NOPAGEHEAD]
                        [;NOEXPANDDIR]
```

Examples:

```
%LISTF @.DEV-TEST@-@KEY, 0
%LISTF @.@(CODE="EDTCT"), 1
%LISTF @.@(SECTORS>10000 and NOT ISPRIV), 2
%LISTF @.@(ACCDATE<TODAY-120 and NOT ISPRIV),3; *LP
%LISTF @.DATA(ONDEVICE(2)), 4
%LISTF @.UTIL.SYS(ISPROG and PROG.PMCAP), 5
%LISTF MAIN000.DATA, ACCESS
%LISTF @.DB.@(DBSETFULLNESS>.70 and DBSETTYPE<>"D"), DB
%LISTF @.DATA, DISCUSE
%LISTF @.DATA, SEC
%LISTF MYFILE, XL3
%LISTF @:SPOOL, 2
%LISTF @.@.@:TEMP, 3
```

MPEX's %LISTF command is a vastly extended version of MPE's :LISTF command:

- Naturally, not just MPE filesets but MPEX FILESETS (see some of the examples above) can be %LISTFed;
- Many useful new modes (3, 4, 5, DB, XL3, and more) have been implemented. These modes can show you information that you never even dreamed of getting with MPE :LISTF.
- Several synonyms are available for the numeric modes. For instance, instead of having to remember to use -2 to show information on ACDs, simply use %LISTF MYFILE,ACD instead.
- Tempfilesets and spoolfilesets can also be %LISTFed, using many of these modes, by simply appending :TEMP or :SPOOL to the fileset.
- On POSIX systems, %LISTF will default to listing files from your Current Working Directory (CWD), which may not be an MPE GROUP if you have issued a :CHDIR command.
- Also on POSIX systems, %LISTF will list, where appropriate, files found in subdirectories of the group or directory specified. This behavior can be turned off by using the keyword ;NOEXPANDDIR.

By default, some %LISTF modes output no file/sector totals, while others output file/sector totals only for those groups which have more than one file shown, for those accounts which have more than one group shown, and grand totals only if more than one account is shown. All %LISTF modes (except for mode 6) by default print page headers at the top of each page of output.

There are four keywords that you can put on a %LISTF that can alter these defaults:

- **;NOTOTALS** — don't output any file/sector totals.
- **;NORMTOTALS** — output file/sector totals only for groups with more than one file, for accounts with more than one group, and grand totals only if more than one account.
- **;ALLTOTALS** — output file/sector totals and grand totals for all groups and accounts, even if there is only one file in the group, one group in the account, or one account.
- **;NOPAGEHEAD** — do not output page headers.

All %LISTF modes do a page break every 60 lines. To change the page size, type

```
:SETJCW VESOFTPAGESIZE=numlinesperpage
```

If you have more than one HP3000 and would like to have your system name show in the %LISTF headings, simply do a

```
%SETVAR HPSYSNAME "yoursystemname"
```

(e.g. "%SETVAR HPSYSNAME 'ALPHA'") before doing the %LISTF. Even better, add the SETVAR to your MPEXMGR.PUB.VESOFT file and the system name will always show on all %LISTFs.

Note: HPSYSNAME is also used by SECURITY for the reports generated by LISTLOG and LISTUSERS.

One final note on using %LISTF: if the number of files to be listed is very large (more than 10000), you may get an error message regarding the file SORTWRIT.PUB.VESOFT. If this occurs, set the variable VESOFTSORTFILESIZE to the number of files you expect to be listed.

```
%SETVAR VESOFTSORTFILESIZE 100000
%LISTF @.@.@,DATESORT.LISTF.VECSL
```

%LISTF modes

%LISTF supports several different formats or "modes". Here is a brief listing of each mode, showing the symbolic name and mode number (when available). Several of these mode names correspond to the symbolic names that MPE's :LISTFILE command supports.

Number	Name	Description
-3		HP's mode -3: file label as readable text and LOCKWORD/CREATOR.
-2	ACD	HP's -2 mode: shows ACD information.
-1	LABEL	HP's mode -1: shows file label in hex. (MPE/iX)
0	FILES	Default %LISTF mode: filenames only.
1	SUMMARY	HP's mode 1; limited file information (TYPE, EOF, FLIMIT).
2	DISC	Mode 1 information + disk space usage + file's age.
	XL3	HP's mode 3; file label as readable text.

Number	Name	Description
	DETAIL	HP's mode 3, see XL3 for details.
3		MPEX's mode 3; shows last access/modify/creation and restore dates.
	XL4	HP's mode 4: shows security related information.
	SECURITY	(same as above)
4		MPEX's mode 4: shows file location (full extent map).
5		MPEX's mode 5: shows program information (PREP parms, CAPs).
6	FILENAME	HP's mode 6: fully qualified filenames.
	QUALIFY	(same as above)
	ACCESS	Displays users accessing a file or fileset.
	DATES	Displays access, modify, create, restore, and state change dates.
	DB	Shows DATABASE statistics.
	DISCUSE	Shows drive-by-drive allocation of a fileset.
	ID	Helps identify the type and contents of a file.
	SAVABLE	Displays space savable using ;XLTRIM or ;SQUEEZE for each file.
	SEC	Shows file security considering group and account access.
	POSIX	Shows POSIX-related file attributes.

%LISTF...,0 [FILES]

Syntax:

```
%LISTF fileset [,0]
```

Examples:

```
%LISTF @.DEV-TEST@-@KEY(CODE="EDTCT"), 0
ACCOUNT=  VESOFT      GROUP=  DEV

AP000S      AP000U      AP010S*      AP100P      AP100S      BADRECS*
COMP000      COMP999      ...
```

As you see, MPEX's %LISTF...,0 is quite similar to MPE's :LISTF...,0, but with a few distinct improvements:

- Of course, the full power of MPEX filesets — with "+"s, "-"s, and selection criteria — is supported. This lets you select files in ways that you never could before.
- Account and group names are printed — if you type a command such as

```
%LISTF @CLOCK@.@.@
```

you'll get group and account headers for each group in which a file is found (unlike MPE's :LISTF...,0).

- Files that are in use are marked with an "*" (we thought that since we had the space on the listing, we might as well use it).

%LISTF...,1 [SUMMARY]

Syntax:

```
%LISTF fileset,1 [;listfile]
                [;NOTOTALS|NORMTOTALS|ALLTOTALS]
                [;NOPAGEHEAD]
```

Examples:

```
%LISTF @.DEV-TEST@-@KEY(CODE="EDTCT"), 1

ACCOUNT=  VESOFT          GROUP=  DEV

FILENAME  CODE  -----LOGICAL RECORD-----
              SIZE  TYP          EOF      LIMIT

BEXP      EDTCT   256W  FB          633      958
...

ACCOUNT  TOTAL:    192 FILES                      35722 SECTORS
```

MPEX's %LISTF...,1 is just like MPE's :LISTF...,1 except that:

- It has the full power of MPEX filesets — + filesets, – filesets, selection (by code, by disk space, by record size, etc.).
- File counts and disk space totals (group, account, and grand) are printed.

%LISTF...,2 [DISC]

Syntax:

```
%LISTF fileset,2 [;listfile]
                [;NOTOTALS|NORMTOTALS|ALLTOTALS]
                [;NOPAGEHEAD]
```

Examples:

```
%LISTF @.@(SECTORS>10000 and NOT ISPRIV),2

ACCOUNT=  LIST          GROUP=  VESOFT

FILENAME  CODE  -----LOGICAL RECORD-----  ----SPACE----  --DAYS--
              SIZE  TYP    EOF LIMIT R/B  SECTORS #X MX ACC  MOD

CUST01   *      2560W  FB     566  566   1    11340 32 32   1   1
              current 540    2 readers, 2 writers
...

ACCOUNT  TOTAL:    15 FILES                      231007 SECTORS
```

```
%LISTF ./A@/,2

PATH= /DOC/WORK/

FILENAME CODE  -----LOGICAL RECORD-----  ----SPACE-----  --DAYS--
              SIZE  TYP   EOF LIMIT R/B   SECTORS #X MX ACC  MOD

A01      KSAM   128W  FB      0  1023   1       128  1  *  20  20
A01K     KSAMK  128W  FB     210   210   1       224  2  8  20  20
A09      /      ==>  NEWFILE
A16      link   /DOC/WORK/A09/NEWFILE      16  1  *
ACORELFT  80B   FA     135   135  16       48  1  1 119  145
...

DIRECTORY TOTAL:    17 FILES                      16432 SECTORS

PATH= /DOC/WORK/A09/

FILENAME CODE  -----LOGICAL RECORD-----  ----SPACE-----  --DAYS--
              SIZE  TYP   EOF LIMIT R/B   SECTORS #X MX ACC  MOD

NEWFILE      128W  FB      0  1023   1         0  0  *  20  20
```

MPEX's %LISTF...,2 is just like MPE's :LISTF...,2 except that:

- It has the full power of MPEX filesets — + filesets, – filesets, selection (by code, by disk space, by record size, etc.).
- File counts and disk space totals (group, account, and grand) are printed.
- The AGE of the file, both in terms of last access and last modification, are shown (in days).
- If the file is in use, the number of readers, the number of writers, and the actual EOF (if it's different from the one kept on disk) are also output.

Note that on MPE/iX systems, native mode KSAM files (KSAMXL) will almost always report an EOF that is greater than the FLIMIT. This is due to the fact that a KSAMXL file contains both the DATA and the KEY information in the same file. The FLIMIT reported by :LISTF is the maximum number of DATA records, the CURRENT EOF reported by %LISTF shows the number of both KEY and DATA records combined.

- On systems supporting POSIX, the contents of directories are shown as well as the destination of any LINK files.

%LISTF...,3

Syntax:

```
%LISTF fileset,3 [; listfile]
                  [;NOTOTALS|NORMTOTALS|ALLTOTALS]
                  [;NOPAGEHEAD]
```

Examples:

```
%LISTF @.@.(ACCDATE<TODAY-120 and NOT ISPRIV),3

ACCOUNT=  TECH          GROUP=  EUGENE

FILENAME CODE CREATOR CRE-DATE  MOD-DATE  MOD-TIME  ACC-DATE  RST-DATE

ADDTECHJ      EUGENE  31 DEC 90 02 OCT 91 10:05 AM 02 OCT 91 13 DEC 91
AL            EUGENE  06 MAR 91 02 OCT 91 10:05 AM 28 OCT 91 13 DEC 91
ASPLASH       EUGENE  24 OCT 91 12 DEC 91 04:58 PM 12 DEC 91 13 DEC 91
...

GROUP      TOTAL:    41 FILES                      52344 SECTORS
```

%LISTF...,3 can show information that's unavailable elsewhere in any sort of compact format:

- The creator id of the file.
- The creation date of the file.
- The date the file was last modified.
- The TIME the file was last modified (not shown in the above example).
- The date the file was last accessed.
- The date the file was last restored — this is usually equal to the creation date unless you did a **:RESTORE ;OLDDATE** of this file, in which case it is the date of the restore.

These features are particularly useful when used together with the ACCDATE, CREATOR, MODDATE, CREDATE or RSTDATE selection criteria (note that everything that can be shown using MPEX's %LISTF can be selected by as well!).

For instance,

```
%LISTF @.DATA(CREATOR<>"DBA"),3
```

will show you the creator IDs of all the files in the DATA group that weren't created by the user DBA. If you want to get REALLY fancy, you can say

```
%LISTF @.@.(NOT USEREXISTS(CREATOR+"."+ACCOUNT)),3
```

This will find all the files in the system whose creators don't exist. How does it do this? See **"[;CREATOR=](#)"** in the %ALTFILE command discussion — it's too long to repeat here.

Another interesting use for last access date selection criteria is security — which files were accessed over the weekend? If you come in on Monday, you can say

```
%LISTF @.@.(ACCDATE=TODAY-1 or ACCDATE=TODAY-2),3
```

This will find all the files accessed on Sunday (TODAY-1) or Saturday (TODAY-2). The mode 3 listing will tell you which day the access was on, and also whether the file was also modified on that day.

%LISTF...,4

Syntax:

```
%LISTF filespec,4 [;listfile]
[;NOTOTALS|NORMTOTALS|ALLTOTALS]
[;NOPAGEHEAD]
```

Examples:

```
%LISTF @.DATA(ONDEVICE(2)), 4

ACCOUNT=  VESOFT      GROUP=  DATA

-----FILE----- EXTENTS -BLKFACT- -----SECTORS----- DEVICE      LABEL
NAME          CODE  NUM MAX  NOW BEST  USED NOW  SAVABLE  CLASS      LDEV

MYFILE                4  32    1    3      136      28  DISC      2
  Dev/Sector:    1/%0000512114    2/%0000227353    1/%0000512156
  Dev/Sector:    3/%0000725756
MYDATA                7  32    1      238      8  DISC
  Dev/Sector:    3/%0000321710    2/%0000231143    2/%0000255345
  Dev/Sector:    1/%0000512220    2/%0000263320    3/%0000535323
  Dev/Sector:    1/%0000216607
  ...

GRAND  TOTAL:      37  FILES      115227  SECTORS
```

%LISTF...,4 displays information on the disk space used by a file. It shows certain things that :LISTF...,2 doesn't:

- The device class on which the file resides (i.e. ;DEV=... parameter the file was built with).
- The best blocking factor for the file: the blocking factor that optimizes the file's disk space. (Omitted on MPE/iX systems as blocking factor is irrelevant to MPE/iX disk space usage.)
- The total amount of savable disk space: the amount of space you'd save if you %ALTFILEd this file with ;BLKFACT=BEST;SQUEEZE (improving the blocking factor and setting its FLIMIT to its EOF).
- The file extent map: on which device (and where on the device) each file piece resides.

What is the extent map?

A file is not necessarily built as a single contiguous chunk of disk space. It can be divided into up to 32 (on MPE/iX, even more) pieces called "extents".

It's conceivable that every extent in the file is on a different disk drive — in fact, when the :STORE command (on MPE/V) shows you the "device number" on which the file resides, it's only telling you the device on which the first extent resides. It's quite possible that all remaining extents actually reside on other disk drives.

Therefore, before doing any "disk balancing" (for instance, using %ALTFILE...;DEV=... to move IMAGE details onto one drive and masters onto another), you should be certain which devices the file actually resides on.

The %LISTF...,4 listing (and also the %LISTF...,DISCUSE listing, which we'll discuss shortly) can give you this information.

Related MPEX features

As we mentioned, %LISTF...,4 is often useful when you're doing %ALTFILE...;DEV=. On MPE/iX systems, any files that show some value in the "savable disk space" column should be %ALTFILE...;XLTRIMed.

Selection criteria that you might want to use with %LISTF...,4 include:

- **ONDEVICE**(*x*), which selects those files which have at least one extent on device *x*.
- **DEVICESECTORS**(*x*), which returns the number of sectors of the file that reside on the given device —

```
%LISTF @.DATA (DEVICESECTORS(1) >= SECTORS/2), 4
```

will show the extent maps of those files at least half of whose sectors reside on device 1.

- **DEVICE**, a string that contains the file's true device class —

```
%LISTF @.DATA (DEVICE = "SYSDISC"), 4
```

will find all the files built with ;DEV=SYSDISC.

- **BESTBLOCKFACTOR**, the same best blocking factor value displayed by %LISTF...,4;

```
%LISTF @.@(BESTBLOCKFACTOR <> BLOCKFACTOR), 4
```

will find all the files whose blocking factor can be improved.

- **SECTORS**, the number of sectors used by the file.
- **SAVABLESECTORS**, the numbers of sectors savable by reblocking and ;SQUEEZEing (on MPE/V) or ;XLTRIMing (on MPE/iX).

Of course, like on all commands, any fileset with any selection criteria can be used — the ones we just listed are only those that are most applicable to %LISTF...,4.

%LISTF...,5

Syntax:

```
%LISTF fileset,5 [;listfile]
                  [;NOTOTALS|NORMTOTALS|ALLTOTALS]
                  [;NOPAGEHEAD]
```

Examples:

```
%LISTF @.UTIL.SYS(ISPROG and PROG.PMCAP), 5

ACCOUNT=  SYS           GROUP=  UTIL

FILENAME  MAXDATA  STACK  DLSIZE  DBSIZE  #SEGS  CAPABILITIES
          (NMSTACK)    (NMHEAP)

FILEUTIL   8192   1200    600    1231     3  BA IA PM PH [ZERODB]
SLEEPER   DEFAULT  1241     0     111     2  BA IA PM
SOO        20000  1200    768    2696    10  BA IA PM PH
```

%LISTF...,5 shows you all about program files. The most important things here are:

- The program's MAXDATA attribute: how large the program's stack can grow.
- The capabilities which the program has — PM (Privileged Mode) capability is, of course, of particular interest.

Also included are:

- DBSIZE, the size of the program's global variable area;
- #SEGS, the number of segments in the program;
- STACK and DL, the ;STACK=... and ;DL=... values the program was :PREPped with;
- If the program was :PREPped with ;ZERODB, "[ZERODB]" is displayed after the capabilities.
- On POSIX systems, NM program files' NMSTACK and NMHEAP values are displayed instead of the CM MAXDATA, STACK, DLSIZE, and DBSIZE fields. An extra heading line is also supplied to identify the two new fields.

Only program files are displayed — any non-program files in the fileset are simply skipped.

Related MPEX features

%LISTF...,5 is often used in conjunction with the %ALTFILE ;MAXDATA=... and ;CAP=... keywords — you can easily find out what MAXDATA and capabilities a program has, and then change them with %ALTFILE.

The other important use of %LISTF...,5 is finding programs that have the sensitive PM and MR capabilities. PM, of course, should be carefully controlled because it gives a program complete power over the system; MR, if improperly used, can make it easy for deadlocks to occur, which would require a system restart to resolve. For example, you might say

```
%LISTF @.@.@-@.PUB.SYS-@.@.VESOFT&
        (ISPROG and (PROG.PMCAP or PROG.MRCAP)),5
```

This will show you all the programs in the system (except those in PUB.SYS and the VESOFT account) that use either PM or MR capability.

%LISTF...,6 [FILENAME, QUALIFY]

Syntax:

```
%LISTF fileset,6 [;listfile]
                  [;NOTOTALS|NORMTOTALS|ALLTOTALS]
                  [;NOPAGEHEAD]
```

Examples:

```
%FILE FILELIST;TEMP;REC=-26,,F,ASCII;NOCCTL;DISC=99999
%LISTF @.@.@(ACCDATE<TODAY-90 and NOT ISPRIV),6; *FILELIST

(FILELIST will contain:
AP000S.DEV.AP
AP001S.DEV.AP
...
)
```

"%LISTF *fileset*,6" will output (to \$STDLIST or to the specified destination file) the fully-qualified filenames of all the files in the given fileset.

MPEX COMMAND REFERENCE: %LISTF

This can be very useful for generating indirect files to be used by :STORE or any other program that asks for a list of fully-qualified filenames. For example, our %MPEXSTORE command looks something like:

```
%FILE FILELIST;TEMP;REC=-26,,F,ASCII;NOCCTL;DISC=99999
%LISTF [A-Z]@.@.(ACCDATE<TODAY-90 and NOT ISPRIV),6; *FILELIST
%STORE !FILELIST; *T; SHOW
```

will store all the non-database files that haven't been accessed in the past 90 days.

The :STORE command can't handle an MPEX fileset (with its selection criteria), but it can take an indirect file (a file containing filenames). We then use MPEX's %LISTF...,6 to create an indirect file and then feed this indirect file to the :STORE command.

Note, however, that if you want to use %LISTF...,6 to create such an indirect filename list, you must specify ;NOCCTL on the file equation for the list file.

%LISTF..., -1 [LABEL]

Syntax:

```
%LISTF fileset, -1 [; listfile]
                        [;NOTOTALS|NORMTOTALS|ALLTOTALS]
                        [;NOPAGEHEAD]
```

Examples:

```
%LISTF MYFILE, -1
```

%LISTF..., -1 is very similar to the MPE :LISTF..., -1 (which shows the exact contents of a file label) except that MPEX's %LISTF can work on MPEX filesets.

To do a %LISTF..., -1 of a file, you must either have SM capability or have AM capability in the file's account.

%LISTF..., -1 is included mostly for compatibility and completeness; typically, all information you need to know is available far more clearly from one of the other %LISTF modes.

%LISTF..., -2 [ACD]

Syntax:

```
%LISTF fileset, -2 [; listfile]
                        [;NOTOTALS|NORMTOTALS|ALLTOTALS]
                        [;NOPAGEHEAD]
```

Examples:

```
%LISTF @.DATA-AP@.DATA(ACCESSIBLE("R")), -2

FILENAME          -----ACD ENTRIES-----
CONTROL           @.PROD                      : R
                  JACK.PROD                    : R,W,X,A,L,RACD
                  LEARN.PROD                    : NONE
                  OPERATOR.SYS                  : R,RACD
DATATEST          NO ACDS
D770UPD           NO ACD ACCESS
...
```

```
%LISTF @.@.(HASACD), -2
    << shows ACD information on all files in the system that >>
    << have an ACD >>
```

%LISTF..., -2 shows the Access Control Definitions (ACDs) of files in an MPEX fileset.

The output of %LISTF..., -2 is just like the output of MPE's :LISTF..., -2 except that, of course, MPEX %LISTF lets you use MPEX filesets, with selection conditions (such as ACCESSIBLE("R"), which means "all files that I can read"), + filesets, and - filesets.

Just like MPE :LISTF..., -2, MPEX shows "NO ACDS" when a file has no ACD and "NO ACD ACCESS" when a file has an ACD but you are not authorized to see it.

%LISTF..., -3

Syntax:

```
%LISTF fileset, -3 [; listfile]
                    [; NOTOTALS | NORMTOTALS | ALLTOTALS]
                    [; NOPAGEHEAD]
```

Examples:

```
%LISTF MYFILE, -3

*****
FILE MYFILE.DEV.VESOF

FILE CODE : 1029          FOPTIONS: BINARY, FIXED, NOCCTL, STD
BLK FACTOR: 1            CREATOR  : MANAGER
REC SIZE: 256 (BYTES)    LOCKWORD: MYLOCK
BLK SIZE: 256 (BYTES)    SECURITY--READ: ANY
EXT SIZE: 1529 (SECT)    WRITE: ANY
NUM REC: 1528            APPEND: ANY
NUM SEC: 1529            LOCK: ANY
NUM EXT: 1               EXECUTE: ANY
MAX REC: 1528            **SECURITY IS ON
MAX EXT: 1               FLAGS   : n/a
NUM LABELS: 0            CREATED  : TUE, JAN 26, 1991
MAX LABELS: 0            MODIFIED: TUE, JAN 26, 1991, 1:52 PM
DISC DEV #: 2            ACCESSED: TUE, JAN 26, 1991
CLASS      : DISC        RESTORED: TUE, JAN 26, 1991, 1:35 PM
SEC OFFSET: 1            LABEL ADDR: %00100610103
```

%LISTF..., -3 is exactly like %LISTF..., XL3 except that it shows the lockwords (if any) of the files listed, and it lets you use MPEX filesets.

Needless to say, this mode can only be used by users with SM capability or by users with AM capability listing files in their own accounts.

```
%LISTF filesset,ACCESS [;listfile]  
[;NOTOTALS|NORMTOTALS|ALLTOTALS]  
[;NOPAGEHEAD]
```

%LISTF MYFILE@(ISASCII), ACCESS							
ACCOUNT=		PROD		GROUP=		DATA	
FILENAME	---TYPE--	JOBNUM	PIN	----PROGRAM----	ACCESS	RECORD	-LOCK-
MYFILE	OPENED	#S141	16	AP000.PUB.PROD	INPUT	16	
				tim,clerk.payroll,pub (ldev 24)			
	OPENED	#J478	39	AP090.TEST.PROD	APPEND	127	
				update,mgr.dev,test			
	OPENED	#S193	55	AP000.PUB.PROD	INPUT	18	
				laura,clerk.payroll,pub (ldev 37)			

- What they're doing to the file — opening it, running it, storing it, restoring it, etc.;
- Who they are — their job/session numbers, logon IDs, program names, and PINs;
- How they're accessing it (INPUT, OUTPUT, APPEND, etc.);
- Where they are in the file (the record number of the most currently read record) — very useful if somebody is reading a file sequentially and you want to see how far they've gotten.

- Whether or not the user is locking the file or perhaps waiting on a file lock (in which case it'll tell you what order it is in the wait queue!).

There are many applications for this sort of capability. Some common ones include:

- You want to access the file exclusively (purge it, rename it, modify it, restore over it, store it, etc.) and the system says that somebody else has it open. Who is it? If you only knew, you could just get user to close it — ask to exit the program for a moment — with `%LISTF...,ACCESS`, you can do this.
- You're running a big batch serial read and you're getting impatient. How far has it gotten? Just do a `%LISTF...,ACCESS` and look at its current record number! (This is also quite useful for checking the progress of your batch compiles.)

- You're using MPE file locking on MPE or KSAM files and sessions are beginning to hang. Who's got the file locked? Maybe they're at a terminal read and have gone to lunch! %LISTF...,ACCESS will tell you who the locker is, who the waiting processes are, in what order the processes are in the wait queue.

Databases

Before TurboIMAGE, %LISTF...,ACCESS worked just fine with databases — databases and their datasets are, after all, just plain MPE files; what works on files will work on databases, too. (%LISTF...,ACCESS will also work very well on TurboIMAGE/XL.)

Unfortunately, TurboIMAGE/V uses a special mechanism called "global FOPEN": instead of having each database user open each dataset that he's using, each dataset is opened once (or, sometimes, 3 times) by the system.

What can %LISTF...,ACCESS do? It can search the entire system for accessors of the dataset and not find anybody in particular, since no specific process actually has the dataset FOPENed. For a dataset, all that it can display is that it is "GLOBALLY OPENED" and the current record number in the dataset (useful only if only one process is accessing that dataset).

For a database root file, things are better, because a root file is opened by each accessing process. If you do a

```
%LISTF APBASE,ACCESS
```

against a root file, you'll see the vital statistics (session number, logon ID, program name, and PIN) of every database accessor (sometimes more than once, since IMAGE may issue more than one FOPEN per process against the same root file).

If you're using TurboIMAGE/V, we recommend you just do a DBUTIL >>SHOW USERS and/or >>SHOW LOCKS. They're specifically tailored for IMAGE databases, and work quite well for them; MPEX's %LISTF...,ACCESS does roughly the same thing for MPE files and KSAM files.

As we mentioned before, %LISTF...,ACCESS will again give you a lot of useful information on MPE/iX machines with TurboIMAGE/XL. It's only for TurboIMAGE/V that %LISTF...,ACCESS's output is somewhat restricted.

Performance notes

As we mentioned before, %LISTF...,ACCESS is in some respects the most internally complicated of MPEX's commands. MPE does not make it easy for us to do a %LISTF...,ACCESS; we have to read every open file entry in the stack of every process to find out which processes have the file open.

This can take a substantial amount of time. %LISTF...,ACCESS is much faster than in some previous versions, but it could still take up to a minute or so on a heavily loaded system. It will not lock down any resources, but it will take some time and processing power.

To optimize %LISTF...,ACCESS, we try to read as much data as possible from each data segment we look at (to avoid having to look at the data segment again later); to do this, though, we need a good deal of stack space.

MPEX COMMAND REFERENCE: %LISTF

If you get a stack overflow abort while doing a %LISTF...,ACCESS, you can tell MPEX to decrease the size of each chunk that it reads from a data segment, thus slowing the command down a bit but possibly avoiding the stack overflow.

You can control the chunk size by saying

```
:SETJCW MPEXLISTFACCESSCHUNKSIZE=number
```

The default value is 4096; you can (without too badly affecting performance) set this to 2048 or even 1024 if necessary; this can save enough stack space to avoid stack overflows.

%LISTF...,ACCESS stack overflows should, in any event, be rare; they most frequently happen when the %LISTF...,ACCESS is being done from within a UDC or as a result of a %REDO (or %DO) command, since UDCs and %REDOs already require a good deal of stack space for themselves.

%LISTF...,DATES

Syntax:

```
%LISTF fileset,DATES
```

Examples:

```
%LISTF MPEX@,DATES
```

```
SYSTEM BATMAN TOM,MANAGER.VESOFT,PUB TUE, NOV 30, 1993, 9:48 AM
```

```
ACCOUNT= VESOFT GROUP= PUB
```

File	Created	Modified	Accessed	Restored	State-Change
-----	-----	-----	-----	-----	-----
MPEX	1/20 11:15	Wed 11:15	Today 9:36	Tue 11:15	Wed 11:15
MPEXMGR	Wed 12:20	Wed 12:20	Today 9:37	Wed 12:20	Wed 12:20

MPEX %LISTF...,DATES is an updated and easier-to-read version of MPEX's %LISTF...,3. This %LISTF mode uses the current setting of the %SET DATE command to determine the proper format to display dates over a week old. For dates within the last week, the day of the week is displayed using a three letter abbreviation. However, for dates older than a week but still in the current year, the date is displayed as MM/DD if the date setting is MDY or YMD or as DD/MM for %SET DATE,DMY. For dates prior to the current year, a six-digit number is displayed without slashes, either in MMDDYY, DDMYY or YYMMDD format, depending upon what was set using the %SET DATE command.

On systems that support POSIX, an additional column labelled "State-Change" is displayed showing the files' state-change date.

%LISTF...,DB

Syntax:

```
%LISTF fileset,DB [;listfile]
                [;NOTOTALS|NORMTOTALS|ALLTOTALS]
                [;NOPAGEHEAD]
```

Examples:

```
%LISTF @.DATA(DBSETFULLNESS>.65 and MODDATE>TODAY-90), DB
```

ACCOUNT=		PROD		GROUP=		DATA						
FNAME	SET	NAME	TYPE	LDEV	ENTRY	CAPA-	%FULL	BLK	SECTORS	%BLOCK	SECTRS	
					COUNT	CITY		FCT		WASTED	WASTED	
ADVT01	CUST		M	3	943	1001	94.2%	16	1024	0.7%	7	
ADVT02	SUPPLY		M		1063	1511	70.3%	51	434	0.1%	0	
ADVT03	SALES		D	2	199	210	94.7%	14	256	0.1%	0	
CUST01	CUST		M		9576	13001	73.6%	23	11340	0.2%	22	
CUST02	SUPPORT		D		5618	8004	70.1%	6	13350	12.7%	1699	
CUST09	AUX-CUST		D	1	525	634	82.8%	2	2544	5.6%	143	
CUST10	LINES		D	2	321	476	67.4%	2	1912	0.1%	1	

As you see, %LISTF...,DB shows you a lot of useful information on IMAGE DATABASE DATASETS:

- The dataset's filename and IMAGE dataset name.
- The dataset type (D=detail, M=manual, A=automatic).
- The LDEV the dataset is on, if it is entirely on a single LDEV; otherwise blank.
- The number of entries in the dataset, the dataset capacity(which is maximum, if DDX is enabled) and the fullness of the dataset.
- The dataset blocking factor (number of records per IMAGE block).
- The number of sectors in the dataset.
- The fraction of space that is wasted in each block (because the total size of the data in each block [including IMAGE pointers] is not an exact multiple of 128 words).
- The total number of sectors wasted because the total data size in each block is not an exact multiple of 128 words.

The major advantages of this mode are the obvious ones:

- You can now see the status of all the databases in your system (or in a particular fileset).
- You can select datasets by various things, especially their fullness:

```
%LISTF @.@@.(DBSETFULLNESS>.70 and DBSETTYPE<>"D"), DB
```

will find all the master (DBSETTYPE<>"D") datasets in the system that are more than 70% full (since access speed to highly-loaded master datasets can be pretty bad).

- You can easily see the percentage fullness and the blocking factor of each dataset.

Related MPEX features

Certain selection criteria fit very nicely with %LISTF...,DB:

- Selection by DBSETFULLNESS (see the example above);
- Selection by DBSETTYPE:

```
%LISTF @.@.(DBSETTYPE<>"D"), DB
```

finds all the non-detail (i.e. manual or automatic) datasets. As you see, DBSETTYPE is always one-character string — "D" for detail, "M" for manual, and "A" for automatic.

- Selection by DBSETNAME, used if you want to show information on a specific dataset:

```
%LISTF BASEAP@(DBSETNAME="INVOICES"), DB
```

Similarly, you might even say

```
%LISTF BASEAP@(DBSETNAME MATCHES "INV@"), DB
```

to show information on all datasets whose names start with INV. This way you can avoid having to figure out the dataset's MPE file name (e.g. BASEAP09).

- If you want to, you can also select on DBSETENTRIES (current number of entries), DBSETCAPACITY (capacity), DBSETBLOCKFACTOR (dataset blocking factor), DBSETENTRYLENGTH (number of words per IMAGE record), and DBSETBLOCKWASTAGE (the fraction of space wasted because the block size is not a multiple of 128 words).

MPEX can get database information on any database (except for those that are exclusively opened), subject to the normal MPE file system security restrictions.

Notes on DBOPEN modes

Unfortunately, some DBOPEN modes are incompatible with one another — if, for instance, you open a database in mode 1, you can't then open it in mode 2 (until the original opener closes it), and vice versa.

Although %LISTF...,DB doesn't use DBOPEN to look at the databases, its FOPEN call is roughly equivalent to a DBOPEN mode 6 (which is compatible with modes 2, 4, and 8, but not with modes 1 and 5). If, however, this fails because the database is already opened in mode 1 or 5, %LISTF...,DB is smart enough to try opening the database the other way, which is compatible with modes 1 and 5.

If MPEX opens the database when it is not opened by anybody else, and then somebody else tries to open it in mode 1 or 5, the would-be opener will get an IMAGE error -1, "DATA BASE OPEN IN AN INCOMPATIBLE MODE" — MPEX didn't know that somebody would try to open the database in mode 1 or 5, so it opened it in mode 6. (This will only happen until MPEX closes the database, which it will do when MPEX starts listing the next database in the fileset or when the %LISTF...,DB finishes.)

If you are concerned that this might happen, and you only open your databases in modes 1 or 5, you can set the MPEXLISTFDBMODE5 JCW to 1 (perhaps in your MPEXMGR.PUB.VESOFT file). This tells %LISTF...,DB to open databases in a way analogous to a DBOPEN mode 5 (read only, compatible with modes 1 and 5, but incompatible with the other modes).

%LISTF...,DISCUSE

Syntax:

```
%LISTF fileset,DISCUSE [;listfile]
                        [;NOTOTALS|NORMTOTALS|ALLTOTALS]
                        [;NOPAGEHEAD]
```

Examples:

```
%LISTF @.DATA, DISCUSE
```

ACCOUNT=	VESOFT	GROUP=	DEV			
FILENAME	CODE	REC	TYPE	DEV 1	DEV 2	DEV 11
ACC	171	508B	VA	50		
BEXP		256W	FB	420	840	60
BHOOKU		256W	FB	40	80	
BPROCSRL	PROG	128W	FB			795
BRLMPE		256W	FB	120	280	
CADDRL		72B	FA	2		
CCALC		72B	FA		10	
CCONVEXP		256W	FB	60	30	
...						
GROUP TOTAL				12317	9422	1529

MPEX's %LISTF...,DISCUSE is an invaluable tool if you care about which files are on which disk drives. For each file in the fileset, it shows you how many of its sectors are on each device; it also prints group, account, and grand totals, for each device.

Some examples of possible uses are:

```
%LISTF BASEAP@.DATA.AP, DISCUSE
```

to show where each of your BASEAP datasets is located; or,

```
%LISTF @.@@(ONDEVICE(1)), DISCUSE
```

to show all the files that are at least partly on the system disk (ONDEVICE(1)) — the DISCUSE listing will tell you exactly how much of each file is on the system disk and how much is on other disks.

%LISTF...,ID

Syntax:

%LISTF *fileset*, ID

Examples:

%LISTF @.PUB.VESOFT, ID

```

MPEX %LISTF @.PUB.VESOFT    PAGE 1
SYSTEM DEVELOP TOM,MANAGER.VESOFT,WORKTE    FRI, NOV  5, 1993,  3:42 PM

FILENAME  CODE      REC  TYPE  EOF  SECT DAYS SINCE
          SIZE                                ACC  MOD

BASIC     PROG      128W FB      360   368  25  25  HOOKed BASIC.PUB.SYS
BYEUDC                                72B  FA      13    16  25  25  LOCKOUT
                                           OPTION LOGON
                                           COMMENT  This UDC
CI         PROG      128W FB      11    16   1  25  Command Interpreter
CIMGR                                256B FA       3    16   1  25  :INITUDCS
                                           :INITMPEXMGR
                                           :INITREDO
DATAFILE                                256B VA       7   256
                                           Encrypted Today at 15:23
                                           by TOM,MANAGER.VESOFT
EDITOR     PROG      128W FB      307   320   1  25  HP32201
FONEDB     PRIV      128W FB        6    16  73  73  Telephone database
FONEDB01   PRIV      512W FB       35   144  73  73  NOTES (master)
FONEDB02   PRIV      384W FB        6    32  73  73  A-KEY (auto)
FONEDB03   PRIV      512W FB       10    48  73  73  PHONES (detail)
FONEDBVE                                72B  FA       2    16  73  73  $user @.@
                                           $program query.pub
                                           $class 1
QEDITMGR   QEDIT      80B TEXT      31    16          20  /set addcurrent
                                           /set check delete
                                           /set decimal
QEDJUMBO   QEDIT    1000B DATA       5    16  23  31  This is an example of
                                           a QEDIT "JUMBO" file
                                           (longer records,etc.)
...

```

In addition to the basic %LISTF...,2 information shown for each file, this %LISTF mode displays extended descriptions or a portion of the file to help you to easily identify each file. This identifying information is retrieved from within files without modifying their last-access date and time.

Note that for QEDIT files, as shown in the above example, the **SIZE**, **TYPE**, and **EOF** columns display more descriptive information. The **SIZE** is the actual record size, the **TYPE** is the internal file type, and the **EOF** is the actual number of logical lines in the file.

For "jumbo" QEDIT files, (files that can have more than 65,535 lines and/or records wider than 256 characters), we add a "+" after "QEDIT" in the **CODE** column.

For files that have been encrypted using the %SEC ENCRYPT command, a two-line display is generated showing the date, time, and user ID of the person who encrypted the file. Note that a specific description created using the %IDENTIFY command will override this display.

If you have used %IDENTIFY to create an extended description for the file, then the extended description you have applied to this file will be displayed.

Although the description that you create using %IDENTIFY may be up to 80 characters, descriptions longer than 31 characters will "wrap around" to the next line on an 80 column display (of course, if you send the listing to the line printer, then the line can be up to 78 characters before it wraps around to the next line).

If you have not added your own description of the file and the file is a dataset within a DATABASE, then information about the dataset will be displayed, such as the name of the set and whether or not it is a master or detail dataset. For CM programs without an extended description, the name of the outer block will be displayed if it is defined. For all other files, the contents of the first three lines of the file will be displayed.

A useful application of the %IDENTIFY command and this %LISTF mode is to use a "keyword" when describing your files. You can then use the IDCONTAINS() function or the %FILTER command to limit the listing to only those files related to a particular command or application.

Here are some useful examples:

```
%LISTF @.PUB.SYS(IDCONTAINS("SQL")),ID
```

Shows all SYSTEM files related to the SQL subsystem.

```
%LISTF @.@.VESOFT(IDCONTAINS("Obsolete")),ID
```

Shows all files in the VESOFT account that are OBSOLETE.

```
%FILTER "LISTF @.MYBASE,ID",DELIM "AP"
```

Shows all files with the keyword "AP" in their description. The ";DELIM" keyword avoids descriptions like "CAPTURED TEXT".

%LISTF...,SAVABLE

Syntax:

```
%LISTF fileset,SAVABLE [;listfile]  
                        [;NOTOTALS|NORMTOTALS|ALLTOTALS]  
                        [;NOPAGEHEAD]
```

Examples:

```
%LISTF @.@.@, SAVABLE
```

```
Group DATA      .DEV      has 10256 savable sectors out of 25984  
  Group USL      .DEV      has 42240 savable sectors out of 59680  
Account          DEV      has 52496 savable sectors out of 85664  
...  
Fileset          has 274224 savable sectors out of 986240
```

%LISTF..., SAVABLE shows you how much disk space can be saved in a fileset (usually @.@.@, meaning your entire system) by using

```
%ALTFILE fileset;XLTRIM
```

on MPE/iX systems. The savable disk space is shown by group, by account, and for the entire system — we show you both the savable and the total space.

This is especially useful in MPE/iX systems because %ALTFILE...;XLTRIM (available only on MPE/iX systems) is so safe — its only effect is to save the space used by the file, without making the file inaccessible in any way and without restricting its further growth. You might very well want to say

```
%ALTFILE @.@.@(SAVABLESECTORS>0 and NOT OPENED);KEEPAMDATES;XLTRIM
```

to trim all the files on your system that are using more disk space than necessary.

%LISTF...,SEC

. Syntax:

```
%LISTF fileset,SEC [;listfile]
                    [;NOTOTALS|NORMTOTALS|ALLTOTALS]
                    [;NOPAGEHEAD]
```

Examples:

```
%LISTF @.DATA, SEC
```

ACCOUNT=	VESOFT	GROUP=	DATA				
FILENAME	CODE	TYPE	READ	APPEND	WRITE	LOCK	EXECUTE
LOG20		FBO	!!!	!!!	!!!	!!!	RELEASED!
LOGOFF		FA	ANY	ANY	ANY	ANY	ANY
MEMOFORM		FA	ANY	AC	AC	AC	AC
VEOPNTS	EDTCT	FA	AC	AC	CR	CR	CR
VEOPNTVE		FA	ANY	AL,GU	CR	AL,GU	AL,GU

%LISTF...,SEC shows information on file security. It shows both the :RELEASE/:SECURE status (in the above example, LOG20 is released — all the other files are secured) and the security matrix of the file (set by :ALTSEC).

As you may know, you can separately allow READ, APPEND, WRITE, LOCK, and EXECUTE access to a file. By default, anybody can do anything to a file (subject to the security attributes of the file's group and account) — however, the :ALTSEC command can be used to change this. For instance, the VEOPNTVE file shown above must have been :ALTSECed by a command such as:

```
:ALTSEC VEOPNTVE;(R:ANY;A,L,X:AL,GU;W:CR)
```

and is, in fact, a probable security breach.

(Files with the same name as the root file of a database but ending with "VE" are usually used by our VEOPEN routine to protect database access. See the SECURITY manual for details on the use of VEOPEN and access control files.)

Normally, the information %LISTF...,SEC shows can only be obtained from LISTDIR5's >LISTF or >LISTSEC commands (or MPE/iX's :LISTF...,3 and :LISTF...,4). However, %LISTF...,SEC is the only command that shows the data in a one line-per-file format (much easier to read than MPE/iX :LISTF's print-outs). Furthermore, %LISTF...,SEC merges the file security matrix, the group security matrix, and the account security matrix, thus showing the true effective security of the file.

Sometimes, the information reported by %LISTF...,SEC will appear slightly different from that shown by LISTDIR5 or MPE/iX :LISTF. For instance, LISTDIR5 might say that read access on a file is allowed to "AC,GU,CR" — %LISTF...,SEC will show this simply as "AC".

This is because AC (account) access implicitly allows GU and CR access, too; thus "AC,GU,CR" is redundant, and is equivalent to just "AC". Although %LISTF...,SEC's and LISTDIR5's outputs appear different, they're really identical.

Related MPEX features

%LISTF...,SEC works closely with:

- The %RELEASE and %SECURE commands — if, for instance, %LISTF...,SEC shows you a lot of released files in a group, you can say

```
%SECURE @.DATA(ISRELEASED)
```

which will :SECURE all the released files in the DATA group.

- The %ALTSEC command can :ALTSEC entire filesets, as well — after looking at a %LISTF...,SEC listing, you might decide to :ALTSEC the entire fileset:

```
%ALTSEC @.DATA;(R:ANY;A,W,L,X:GU,CR)
```

- The SECURITYANY, SECURITYAC, SECURITYAL, SECURITYGU, SECURITYGL, and SECURITYCR selection criteria — each of these is a string that contains (or doesn't contain) the characters "R", "A", "W", "X" or "L" depending on whether or not this access mode is allowed to this class of user.

For instance, if SECURITYANY="R", this means that the only thing that all users may do is Read the file; the other access types are allowed to more restrictive access classes (e.g. AC users only or GU users only).

For example:

```
%LISTF @.DATA(POS("R",SECURITYANY)=0),SEC
```

will find and list all the files in the DATA group that do not allow R access to ANY. POS("R",SECURITYANY) finds the location of the first occurrence of the character "R" in SECURITYANY — if there is no "R", POS returns a 0.

MPEX COMMAND REFERENCE: %LISTF

Thus, `POS("R", SECURITYANY)=0` means "there is no 'R' in SECURITYANY", i.e. Read access is not allowed to ANY.

- The SECURITYR, SECURITYA, SECURITYW, SECURITYL, and SECURITYX selection criteria — each of these is a string that contains (in the same format as in the %LISTF..., SEC listing) the classes of users that are allowed this type of access.

For instance, `SECURITYR="AC"` means that only users in the file's account are authorized to read the file.

%LISTF...,POSIX

. Syntax:

```
%LISTF fileset,POSIX [;listfile]
                        [;NOTOTALS|NORMTOTALS|ALLTOTALS]
                        [;NOPAGEHEAD]
```

Examples:

ACCOUNT= VESOFT		GROUP= PUB					
File	Type	UID	Owner	GID	Group ID	ACD	State
-----	-----	-----	-----	-----	-----	-----	-----
CI	FB	108	MANAGER.VESOFT	106	VESOFT		020109 17:14
MX	FB	109	!MGR.DEV	106	VESOFT		090210 7:33
NL	FB	108	MANAGER.VESOFT	106	VESOFT		020109 17:13
SL	FB	1	!MANAGER.SYS	1	SYS	Must	020109 17:18
XL	FB	108	MANAGER.VESOFT	106	VESOFT		020109 17:15
GROUP	TOTAL:	5	FILES	5	MEGABYTES	20016	SECTORS

%LISTF...,POSIX shows all POSIX-related fields, including:

- Fully-qualified File Owner and corresponding numeric UID value.
- File Group and corresponding numeric GID value.
- ACD state: "Yes" if present, "Must" if mandatory.
- State-change timestamp.

The display also flags the following:

- "?" instead of UID or GID, if the fields are zero, indicating that the corresponding owner or group do not currently exist.
- "!" indicates that the file is located in a different account than its owner.

%LISTF...,XL3 [DETAIL]

Syntax:

```
%LISTF fileset,XL3 [;listfile]
                        [;NOTOTALS|NORMTOTALS|ALLTOTALS]
                        [;NOPAGEHEAD]
```

Examples:

```
%LISTF MAIN,XL3

*****
FILE: MAIN.PUB.VESOFT

FILE CODE : 1030          FOPTIONS: BINARY, FIXED, NOCCTL, STD
BLK FACTOR: 1            CREATOR  : MANAGER.VESOFT
REC SIZE: 256(BYTES)     LOCKWORD: **
                           GROUP ID: VESOFT
BLK SIZE: 256(BYTES)     SECURITY--READ: ANY
EXT SIZE: 561(SECT)      WRITE: ANY
NUM REC: 17926           APPEND: ANY
NUM SEC: 17936           LOCK: ANY
NUM EXT: 34              EXECUTE: ANY
MAX REC: 17926           **SECURITY IS ON
MAX EXT: 32              FLAGS   : 15 ACCESSORS, SHARED, 15 R
NUM LABELS: 0            CREATED  : THU, APR 20, 2000, 8:37 AM
MAX LABELS: 0            MODIFIED: THU, APR 20, 2000, 8:38 AM
DISC DEV #: 1            ACCESSED: MON, MAY 29, 2000, 9:00 AM
                           RESTORED: THU, APR 20, 2000, 8:37 AM
                           STATE   : THU, APR 20, 2000, 8:38 AM
SEC OFFSET: 0            LABEL ADDR: **
UFID      : $055E0001 $46112208 $006924C8 $378224E8 $25E6B9F1
VOLCLASS  : MPEXL_SYSTEM_VOLUME_SET:DISC
```

The XL3 listing mode is very much like MPE/iX's :LISTF...,3 — of course, its big advantage is that it supports the full power of MPEX filesets.

%LISTF...,XL4 [SECURITY]

Syntax:

```
%LISTF fileset,XL4 [;listfile]
                  [;NOTOTALS|NORMTOTALS|ALLTOTALS]
                  [;NOPAGEHEAD]
```

Examples:

```
%LISTF MYFILE,XL4

FILE: MYFILE.WORK.VESOFT

ACCOUNT ----- READ : ANY
                WRITE :
                APPEND :
                LOCK  : ANY
                EXECUTE : ANY
```

```

GROUP -----  READ : GU
                WRITE : GU
                APPEND : GU
                LOCK : GU
                EXECUTE : GU
                SAVE : GU

FILE -----  READ : ANY          FCODE: 0
                WRITE : AC          **SECURITY IS ON
                APPEND : AC
                LOCK : AC
                EXECUTE : ANY

FOR MANAGER.VESOFT: READ, WRITE, APPEND, LOCK, EXECUTE

```

The XL4 listing mode is very much like MPE/iX's :LISTF...,4 — of course, its big advantage is that it supports the full power of MPEX filesets.

Additionally, rather than just giving you the "raw" information about the file's access mask — e.g. that Write access was granted to AC, AL, and GL — it gives you the file's "true" access information, in this case that Write access is actually granted to AC; after all, since AL and GL are subsets of AC, saying "AC, AL, and GL" would be redundant.

Similarly, if a file is built with Write access granted to AC and Lock access granted to GU only, MPEX's %LISTF...,XL4 will show it as "LOCK:AC". This is because Write access implies both Lock and Append access; if you configure Lock or Append access as being more restrictive than Write access, this configuration won't really take effect.

Thus, our output might look somewhat different from the MPE/iX :LISTF...,4 output; however, we believe that our output is actually more informative.

%LISTF — defining your own %LISTF formats

Syntax:

```

%LISTF fileset, listfgroupfile|^formatfile
        [;listfile]
        [;NOTOTALS|NORMTOTALS|ALLTOTALS]
        [;NOPAGEHEAD]

```

Examples:

```

%LISTF @.@.AP, ^MYLISTF.DATA.PROD; *LP
%LISTF @.@.AP, SUPER    << uses your own file: SUPER.LISTF.VESOFT >>

```

Note: This is one of the most sophisticated features of MPEX. It can be very powerful, but also fairly difficult to use. Beginning users should probably skip this section.

In addition to providing you with many powerful LISTF modes, MPEX lets you define your own. You can set up your own "flexible %LISTF format files" that define how you want the output to look; these files essentially contain little PASCAL-like programs that give you all the power of a high-level programming language.

Most of the syntax elements of these files are described in the Advanced Expression Programming section in Appendix B — this will explain the various operators and statements (like IF/THEN/ELSE, WHILE/DO, VAR, etc.) that you have available. You can also look at the files in the LISTF.VESOF group to see how MPEX implements its built-in %LISTF modes.

Once you write a flexible %LISTF format file, you can use it by specifying its filename (prefixed by a "^") as the *mode* parameter of a %LISTF command:

```
%LISTF @.@.AP, ^MYLISTF.DATA.PROD; *LP
```

You may also put the file in the LISTF.VESOF group — in this case, all you need to do is specify the unqualified file name:

```
%LISTF @.@.AP, SUPER
```

will use the flexible %LISTF file called SUPER.LISTF.VESOF. In fact, all MPEX %LISTF modes are implemented by flexible %LISTF format files in the LISTF.VESOF group — DB.LISTF.VESOF implements the DB mode, DISCUSE.LISTF.VESOF implements the DISCUSE mode, etc.

A word of warning is in order here: when upgrading to new releases of our software, we will not make any attempt to preserve any %LISTF files that you may have created and placed in this group. In fact, it is quite possible that we may eventually choose the same name that you have in order to define a new %LISTF format. In that case, our future installation will overwrite your handiwork. So while it may be convenient to use this group, we do not recommend that you place any files here.

Since MPE filenames can't start with digits or special characters, all the numeric modes (0, 1, 2, etc.) are prefixed with Ns in the LISTF.VESOF group (mode 3 is implemented by N3.LISTF.VESOF). Similarly, the negative modes are prefixed with Ms (mode -1 is implemented by M1.LISTF.VESOF).

How is a flexible %LISTF template file implemented? It's an ordinary EDITOR-format file that contains a sequence of statements of the type described in the Advanced Expression Programming section of Appendix B. The file can therefore define its own local variables, use control structures like IF/THEN/ELSE, WHILE/DO, and so on. More importantly, it can reference all FILE ATTRIBUTE VARIABLES pertaining to the current file, just like you can in selection conditions.

For instance, the N2.LISTF.VESOF file (which implements %LISTF..., 2) contains:

```
WRITELN (FILE:8,
        FMOPENED:2,
        CODE:6,
        FMTRECSIZE:6:'RIGHT',
        ' ', FMTTYPE:4,
        EOF:10,
        FLIMIT:11,
        BLOCKFACTOR:4,
        SECTORS:9,
        ' ', NUMEXTENTS:2,
        IF MAXEXTENTS=0 THEN ' *' ELSE STRWRITE(MAXEXTENTS:3)
        );
```

This is what generates the detail lines of a %LISTF..., 2, e.g.

SL	* SL	128W	FB	9768	65535	1	10084	4	26
----	------	------	----	------	-------	---	-------	---	----

The WRITELN statement takes all the specified file attribute variables and formats them to be output.

The Advanced Expression Programming chapter of Appendix B describes various control structures, but does not mention the most important one: **FOR RFILE IN...** This construct is used in flexible %LISTF files to repeat something for all the files in the fileset. The reason that this is only mentioned here is because this is a special statement that is valid only in %LISTF expression files.

```
FOR RFILE IN MPEXFILESETTOTAL
DO
  BEGIN
    statement;
    ...
    statement;
  END;
```

If you look at the LISTF.VESOF files, you'll find that almost all of them use this statement. It simply means "For each file in the current fileset, do the statements between the 'BEGIN' and the 'END'".

A typical flexible LISTF file might look like:

```
statements to do at the beginning of the %LISTF operation;
MPEXHEADER1:="headerline1";
MPEXHEADER2:="headerline2";
MPEXHEADER3:="headerline3";
FOR RFILE IN MPEXFILESETTOTAL
DO
  BEGIN
    statement;
    ...
    statement;
  END;
statements to do at the end of the %LISTF operation;
```

The MPEXHEADER1, MPEXHEADER2, and MPEXHEADER3 variables are set to the header lines that you want to output for each group (and at the top of each page); the statements between BEGIN and END indicate what is to be printed for each file. MPEX automatically takes care of writing the "ACCOUNT=... GROUP=..." headers, the group/account/grand totals, and the page breaks.

For example, %LISTF...,1 is implemented by N1.LISTF.VES0FT, which is similar to:

```
MPEXHEADER1:="FILENAME  CODE  -----LOGICAL RECORD-----";
MPEXHEADER2:="              SIZE  TYP          EOF          LIMIT ";
MPEXHEADER3:="";
FOR RFILE IN MPEXFILESETTOTAL
DO
  BEGIN
    WRITELN (FILE:8,
              FMTOPEMED:2,
              CODE:6,
              FMTRCSIZE:6: 'RIGHT',
              ' ', FMTTYPER:4,
              EOF:10,
              FLIMIT:11
            );
  END;
```

The header variables are set (MPEXHEADER3 is set to the empty string since the header is only 2 lines long — you should always specify all three variables unless you are using MPEXFILESET, described below); the WRITELN indicates what is to be output for each file.

Several other options are also available. Saying

```
FOR RFILE IN MPEXFILESETHEAD
  SELECT selectioncondition
DO
  BEGIN
    statement;
    ...
    statement;
  END;
```

will do the "*statements*" only for those files that match "*selectioncondition*".

For instance, MPEX's %LISTF...,5 (which only outputs program file information) specifies

```
FOR RFILE IN MPEXFILESETHEAD
  SELECT ISPROG
DO...
```

to select only those files with code="PROG" or code="NMPRG".

Saying

```
FOR RFILE IN MPEXFILESETHEAD
```

instead of

```
FOR RFILE IN MPEXFILESETTOTAL
```

will cause MPEX not to print any group, account or grand totals (of the number of files and of the disk space). Instead, you can specify your own things to be done at every "group break" and "account break" by using ONBREAKBEFORE and ONBREAKAFTER (with ALREADYSORTED).

The full syntax of a flexible LISTF format file

```
(* Comments -- may be inserted anywhere *)
MPEXHEADER1 := "headerline1";
MPEXHEADER2 := "headerline2";
MPEXHEADER3 := "headerline3";
other statements to do before any files are processed;
FOR RFILE IN MPEXFILESET[HEAD|TOTAL]
  SELECT selectioncondition
  SORT|ALREADYSORTED (sortfields)
  FIRST
    BEGIN
      statements to do at the beginning of the fileset;
    END
  PAGEHEAD
    BEGIN
      statements to do at the beginning of each page of output;
    END
  ONBREAKBEFORE 1
    BEGIN
      statements to do at the beginning of each new account
      ("ONBREAKBEFORE 1" means "before breaking on sort item 1",
       which is the account if you use "ALREADYSORTED");
    END
  ONBREAKBEFORE 2
    BEGIN
      statements to do at the beginning of each new group
      (or whatever sort item 2 is, if you don't use "ALREADYSORTED");
    END
  DO
    BEGIN
      statements to do for each file;
    END
  ONBREAKAFTER 2
    BEGIN
      statements to do at the end of each group
      (or whatever sort item 2 is, if you don't use "ALREADYSORTED");
    END
  ONBREAKAFTER 1
    BEGIN
      statements to do at the end of each account
      (or whatever sort item 1 is, if you don't use "ALREADYSORTED");
    END
  FINALLY
    BEGIN
      statements to do at the end of the fileset;
    END;
```

You can specify as many or as few of the above clauses as you like — put together they can do some very powerful things.

A very simple example might be:

```
FOR RFILE IN MPEXFILESETHEAD
DO
  BEGIN
    WRITELN (FILE:8, " ", SECTORS:10, SAVABLESECTORS:10);
  END
ONBREAKAFTER 2
  BEGIN
    WRITELN ("TOTAL SAVABLE DISK SPACE: ", TOTAL(2,SAVABLESECTORS));
  END;
```

For each file, this outputs the name, the number of sectors, and the number of savable sectors (savable, that is, by an %ALTFILE...;XLTRIM). After each group is finished (ONBREAKAFTER 2), it writes the total savable disk space in the entire group.

Note the expression

```
TOTAL(2,SAVABLESECTORS)
```

TOTAL(2, x) returns the sum of x for all files in this group; TOTAL(1, x) returns the sum of x for all files in this account; TOTAL(0, x) returns the sum of x for all files in the fileset.

In general,

```
TOTAL( $n$ ,  $x$ )
```

can be said to be the sum of all ' x 'es since the last N th level break; ' x ' can be any file attribute variable (however, ' x ' can not be a user-defined variable whose value is calculated in the "DO" section).

We could have done the same thing without a TOTAL by saying:

```
VAR SPACE: INTEGER;
FOR RFILE IN MPEXFILESETHEAD
  ONBREAKBEFORE 2
  BEGIN
    SPACE:=0;
  END
DO
  BEGIN
    WRITELN (FILE:8, " ", SECTORS:10, SAVABLESECTORS:10);
    SPACE:=SPACE+SAVABLESECTORS;
  END
ONBREAKAFTER 2
  BEGIN
    WRITELN ("TOTAL SAVABLE DISK SPACE: ", SPACE);
  END;
```

This explicitly initializes a tally variable before each group starts, increments it for each file, and outputs it after the entire group is done.

A useful idiom, incidentally, is `TOTAL (n,1)`. This stands for the total number of files since the last *n*-level break — `TOTAL (2,1)` is the total number of files in the group, `TOTAL (1,1)` is the total number of files in the account, and `TOTAL (0,1)` is the total number of files in the fileset.

Another structure you might want to use is:

```
FOR RFILE IN MPEXFILESET
...
```

(as opposed to "FOR RFILE IN MPEXFILESETHREAD" or "FOR RFILE IN MPEXFILESETTOTAL"). This simply tells MPEX not to print any headers (global, group or account) and not to print any totals, but only do those things that you specify in the FOR statement. For instance,

```
%LISTF fileset, FILENAME
```

(which simply lists the fully-qualified names of the files in the given fileset) is implemented using the file `FILENAME.LISTF.VESOF`, which says:

```
FOR RFILE IN MPEXFILESET
DO WRITELN (FULLNAME);
```

You don't want any headers or totals in `%LISTF..., FILENAME` — you just want the fully-qualified file names; this is what "FOR RFILE IN MPEXFILESET" is useful for.

Of course, even if you use "FOR RFILE IN MPEXFILESET", you can still print your own headers (using `FIRST`, `ONBREAKBEFORE`, `ONBREAKAFTER`, and `FINALLY`). It's just that you'd have to specify the headers that you want explicitly. For instance, using

```
FOR RFILE IN MPEXFILESET
ALREADYSORTED (ACCOUNT:8, GROUP:8)
ONBREAKBEFORE 1
WRITELN ("ACCOUNT = ", ACCOUNT)
ONBREAKBEFORE 2
WRITELN ("GROUP = ", GROUP)
DO
WRITELN ("FILE = ", FILE);
```

outputs "ACCOUNT = *accountname*" before every account's files and "GROUP = *groupname*" before every group's file.

Note the "ALREADYSORTED (ACCOUNT:8, GROUP:8)" — this is necessary to tell MPEX that you should expect the files to be sorted in account and group order. With "FOR RFILE IN MPEXFILESETHREAD" and "FOR RFILE IN MPEXFILESETTOTAL" you don't have to do this, but for "FOR RFILE IN MPEXFILESET" you must specify the "ALREADYSORTED" clause if you'll be using `ONBREAKBEFORE`, `ONBREAKAFTER`, and/or `TOTAL`.

By default, all %LISTF output prints the files in sorted order (by account, then by group, then by file).

You can specify your own sort sequence by saying:

```
FOR RFILE IN MPEXFILESET
  SORT (field1, field2,...fieldN)
DO...
```

(of course, you can also specify SELECT, FIRST, FINALLY, and [especially useful] ONBREAKBEFORE and ONBREAKAFTER). Each *fieldX* parameter indicates what you want to sort it on (the highest-precedence things first). (Note: for SORT to work, you must use MPEXFILESET, not MPEXFILESETHHEAD or MPEXFILESETTOTAL.)

For instance, saying

```
FOR RFILE IN MPEXFILESET
  SORT (ACCDATE, MODDATE)
DO...
```

will sort all the files in the fileset by their last access date (ACCDATE), and within that (for all files with the same last access date) by last modify date.

Saying

```
FOR RFILE IN MPEXFILESET
  SORT (ACCOUNT:8, SECTORS)
DO...
```

will sort first by the account and then by the file's disk space. Note that we said "ACCOUNT:8" — the 8 is the maximum length of the account name. Whenever you sort by a STRING, you must specify the maximum length of the string field (otherwise you'll get an error message).

You can also append a ": 'DESC'" to any of the fields to indicate a sort in DESCending order:

```
FOR RFILE IN MPEXFILESET
  SORT (ACCOUNT:8, SECTORS:'DESC')
```

will sort by account, and within each account, in descending order by the number of sectors (i.e. the biggest files first).

When you do your own SORTing, the ONBREAKBEFORE, ONBREAKAFTER, and TOTAL constructs are especially useful. In general, an

```
ONBREAKBEFORE n
  BEGIN
  ...
  END
```

will be executed before the *N*th field in the SORT list changes — thus, for the example of SORT (ACCOUNT:8, SECTORS:'DESC'), any ONBREAKBEFORE 1 will be executed before the start of each new account.

An **ONBREAKAFTER** *N* will similarly be executed after the *M*th field in the **SORT** list changes. **TOTAL** (*n*, *x*) will sum all the values of *x* since the last change in the *M*th field.

One more example that uses **ONBREAKAFTER**, **SORT**, and **TOTAL**:

```
FOR RFILE IN MPEXFILESET
  SORT (ACCOUNT:8, CREATOR:8)
  ONBREAKAFTER 2
    WRITELN (CREATOR, '.', ACCOUNT, ' created ',
             TOTAL(2,1), ' files, ',
             TOTAL(2,SECTORS), ' sectors');
```

This sorts all the files in the fileset by their account and creator ID; then, for each creator (level-2 break), it outputs the number of files created by that user (**TOTAL**(2,1)) and the number of sectors those files occupy (**TOTAL**(2,SECTORS)). Since only an **ONBREAKAFTER** 2 clause was specified, only the creator-level totals will be output; no information will be output for each file, nor will any account or grand totals be printed.

One final note on using **SORT**(...) in a flexible **LISTF** file: if the number of files to be %LISTFed is very large (more than 10000 files), you may get an error message regarding the file **SORTWRIT.PUB.VESOFT**. If this occurs, you can set the variable **VESOFTSORTFILESIZE** to the number of files you expect to be listed.

So, there it is — the way for you to write your own flexible %LISTF files. As you can see, you can do arbitrary sorts, totals, and formatted outputs; you can define %LISTF modes even more complicated (and useful) than the ones we've predefined.

Perhaps the best way to learn how to write flexible %LISTF files is by looking at the VESOFT-supplied files in the **LISTF.VESOFT** group. The only thing we can warn you about is that some of those files use additional, undocumented constructs that we may not be prepared to release yet to the user community at large — however, most of the stuff that the **LISTF.VESOFT** files do can as easily be done by your own flexible %LISTF files.

%LISTJOB

Syntax:

```
%LISTJOB [userset]
          [;NOPROC]
          [;NOSORT]
          [;SUBMIT]
```

Examples:

```
%LISTJOB;SUBMIT
```

JOB#	STATE	DEV	INTRODUCED	Q	PROGRAM	LOGON
#S27	EXEC	20	TUE 2:28P		MAIN	OPERATOR.SYS,OPERATOR
#S127	EXEC	38	9:44A		QEDIT	CHUCK,MANAGER.TECH,CHUCK
#S116	EXEC	33	7:17A		MAIN	RON,MANAGER.TECH,RON
#S117	EXEC	35	7:37A			RON,same

#J555	EXEC	o766	WED 12:16P	b+MAIN	BACKG,MANAGER.VESOFT,BACKG #s27, operator.sys
#J601	WAIT	D1	THU 10:52A		TEST,MANAGER.TECH,CHUCK #s88, chuck,manager.tech
#J880	SCHED	HI	SAT 12:01A	01/22/94	SCHEDULR,MANAGER.SYS,PUB #j567, schedulr,manager.sys
#A6	SCHED	+	1:30A	-MTWRF-	BACKUP,MANAGER.SYS
#A4	SCHED	+	12:35A	S-----A	DISCLEAN,same
#A8	SCHED	+	WHENEVER (HPDATE=1)		REPORT,same
#A11	SCHED		FRI 10:00A	94/09/16	TESTSCHD,same
#A1	SCHED	+	12:10A	SMTWRF	GETDFREE,MANAGER.TECH
#A2	SCHED	+	12:15A	-MTWRF-	DAILY,MGR.XPRESS
#A7	SCHED	+	WHENEVER BETWEEN(HPDAY,2,6)...		REMOTE,USER.XPRESS
JOBFENCE = 6; JLIMIT = 3; SLIMIT = 33					

The %LISTJOB command is similar to the MPE :SHOWJOB command, but displays additional information and allows for selection based upon a userset.

The key features of this command are:

- Name and queue of the currently running process. If there is more than one process running, the first process on the process tree will be listed with a "+" character to indicate additional processes are active.
- The MPE group name is displayed for each user.
- The output is sorted by job type (sessions first, then jobs), account, user, and finally the job or session number.
- Selection based upon MPEX usersets.

Note that fields that have trivial values — such as the STATE when it's EXEC, the LDEV for jobs, the INTRO date when it's today, etc. — are not displayed.

The ;NOPROC keyword disables the display of the currently executing process for each job or session.

The ;NOSORT keyword disables the sort step prior to display. Without the sort, multiple occurrences of the same user, group and account may not be displayed as "*Sessionname,same*" as shown above.

The ;SUBMIT keyword, when specified, displays additional information about each Job. With this keyword, an additional line indicating the original submitter of the job will be displayed.

At the end of the listing, STREAMX scheduled jobs are displayed as follows:

- For ordinary jobs that do not repeat or have a particular condition to be streamed are shown with the scheduled date and time to submit in the INTRODUCED and PROGRAM columns.
- Jobs that repeat are indicated with a "+" character to the right of the word "SCHED".
- If the job is to be submitted when a particular condition occurs, the conditional expression for that job will be displayed. If the condition is too long to display in the field, then it is truncated and "..." is printed to show that it is too long to display.

MPEX COMMAND REFERENCE: %LISTREDO

- For jobs that repeat on specific days, the days that the job repeats is shown in the PROGRAM column.

This is essentially the same format as the %SHOWJOB command, and shows the same information as the %SEC SHOWSCHED command. %SEC SHOWSCHED, however, will display the entire condition under which a job will be submitted.

%LISTREDO

Syntax:

```
%LISTREDO [abscmdnum1[/abscmdnum2]]    [;OUT=outfile]  
          [-relcmdnum[/-relcmdnum2]]    [;{REL|ABS|UNN}]  
          [""][@]string[""]              ]
```

Examples:

```
%LISTREDO  
%LISTREDO 10/20  
%LISTREDO START=10; END=20  
          (* for compatibility with MPE/iX *)  
%LISTREDO -30;REL  
%LISTREDO RUN  
%LISTREDO @MYFILE  
  
%FILE LP;DEV=LP  
%LISTREDO;OUT=*LP;UNN
```

%LISTREDO displays those commands that you entered into MPEX and that have been saved in the MPEX command history.

%DO, %LISTREDO, and %REDO are related, and are discussed in full detail under the %REDO command in this manual.

%MANY

Syntax:

```
%MANY delimiter mpexcommand [delimiter...]
```

Examples:

```
%MANY \PURGE DATAFILE\BUILD DATAFILE\RUN PROGFILE  
%MANY %REPEAT %BUILD FILE!N %FORNUM N=1,10
```

The %MANY command lets you specify several MPEX commands on one line; for instance,

```
%MANY %REPEAT %RUN MYPROG;INFO="!MPEXCURRENTFILE" %FORFILES A@
```

will execute the given %REPEAT...%FORFILES loop just as if you'd typed

```
%REPEAT  
%  RUN MYPROG;INFO="!MPEXCURRENTFILE"  
%FORFILES A@
```

This can be especially useful for "\$" commands (to submit a job stream that executes several commands) and for `:RUN MPEX;INFO="..."`.

Note that the commands are delimited by the first character in the %MANY command's parameter list — the character "%" was used above, but if you wanted to, you could have used something else, e.g.

```
%$MANY ^REPEAT ^RUN MYPROG;INFO="!MPEXCURRENTFILE" ^FORFILES A@
```

Just make sure you use a character that doesn't occur in any of the commands you are going to enter!

%MPEXSTORE

Syntax:

```
%MPEXSTORE fileset; storeparms
```

Examples:

```
%MPEXSTORE @.@.@-@.@.SYS-@.@.VESOFT &
          (ACCDATE<TODAY-90 and NOT ISPRIV);*T;SHOW
```

The MPE `:STORE` command can already handle filesets — however, its filesets are not nearly as flexible as MPEX filesets. In particular:

- It doesn't have selection criteria such as `CODE`, `SECTORS`, `CREATOR`, etc. For instance, you can't store all the files that haven't been accessed in 90 days EXCEPT FOR databases — `:STORE` can't select by filecode.

The %MPEXSTORE command lets you use the full power of MPEX filesets with MPE `:STORE`. For instance, the command

```
%MPEXSTORE @.@.@-@.@.SYS-@.@.VESOFT &
          (ACCDATE<TODAY-90 and NOT ISPRIV);*T;SHOW
```

will store all files that haven't been accessed in 90 days EXCEPT:

- databases (ISPRIV),
- files in the SYS account (-@.@.SYS), and
- files in the VESOFT account (-@.@.VESOFT).

Of course, all the other fileset features and selection criteria are available, too — you can store all the files created by a particular user, store all the files except for files with code PROG and USL (since in the event of failure they could easily be regenerated), etc.

This command works by doing %LISTF..., FILENAME to write the fully qualified names of all the files in the fileset to a disk file; then, it uses the disk file as an indirect file for MPE `:STORE`. For large filesets, this means that it might take some time for this command to start up (since it has to go through the entire fileset before doing the `:STORE`). Please be patient...

%NEWLINK

Syntax: `%NEWLINK linkfileset, targetfileset`

Examples:

```
%NEWLINK MYLINK, REALFILE
%NEWLINK WORK, /DEVELOP/PUB/source/c
%NEWLINK =.TEST, @.PROD
```

The %NEWLINK command allows you to create a symbolic link to another file or directory. Since symbolic link files are a POSIX related feature, this command is only available on machines that support POSIX.

Symbolic link files are files that "point" to other files, somewhat like a permanent file equation. When a link file is referenced in a command or program, the file that the link points to is actually opened. Our %LISTF...,2 shows the target of a link file as follows:

```
%LISTF ./A@/,2

PATH= /DOC/WORK/

FILENAME  CODE  -----LOGICAL RECORD-----  ----SPACE----
          SIZE  TYP      EOF    LIMIT R/B  SECTORS #X MX

A09      /      ==>  NEWFILE
A16      link  /DOC/WORK/A09/NEWFILE      16  1  *

PATH= /DOC/WORK/A09/

FILENAME  CODE  -----LOGICAL RECORD-----  ----SPACE----
          SIZE  TYP      EOF    LIMIT R/B  SECTORS #X MX

NEWFILE   80B  FA      1420    1420  16      448  *  *
```

In this example, the file A09 is a DIRECTORY containing the file NEWFILE. The file A16 is LINKED to NEWFILE, so if you were to reference A16 in your command (such as :VI A16 to invoke the VI editor), then the file /DOC/WORK/A09/NEWFILE will be opened as the actual work file.

This command is quite a bit different than other commands that have a "*source*" and "*target*" fileset. For this command, the files in the linkfileset are based upon files in the targetfileset, rather than the target files being based upon the source files. The third example above shows how to create a link file in the TEST group for each file that exists in the PROD group. This would allow you to perform a test against "live data" without having to move files from one location to another.

In order to remove a symbolic link file, the command %PURGELINK must be used instead of the regular %PURGE command. (see the [%PURGELINK command](#) for details)

%NOMSG

Syntax: `%NOMSG command`

Examples: `%NOMSG INPUT ANSWER; PROMPT="OK to continue? "; WAIT=5`
`%REPEAT`
`...`
`%NOMSG FORFILES A@.SOURCE`

Saying

`%NOMSG command`

does the same thing as saying

`%command`

(without the **NOMSG**) except that **%NOMSG** inhibits the printing of any MPE CI or file system errors associated with the command.

The **CIERROR** and **FSERROR** JCWs are still set; if the command encounters a fatal error and no **%CONTINUE** is in effect, the command file or UDC will still be flushed. The only difference is that no error message will be printed.

This can be quite useful in cases where you expect an error to occur, and don't want to burden the user with an irrelevant warning or error message. For instance, if you have a command file that uses a particular work file and you want to make sure the file doesn't already exist, you might say

`%NOMSG PURGE WORKFILE;YES`

This way, the "FILE NOT FOUND, NO PURGE DONE" warning will not be printed out if **WORKFILE** doesn't exist. Remember that you can still check to see if an error occurred by looking at the **CIERROR** and **FSERROR** variables.

%OPTION

Syntax: `%OPTION [LIST]
[NO LIST]
[RECURSION]
[NO RECURSION]`

Examples: `%OPTION LIST`
`%OPTION RECURSION`

MPEX COMMAND REFERENCE: %PAUSE

MPE/iX lets you have an **OPTION** line not just at the beginning of a command file or a UDC, but also anywhere inside one. The only four options available on this **OPTION** command are:

- **LIST** and **NOLIST**, which turn on or off echoing of the remaining commands in this command file.
- **RECURSION**, which makes MPEX search for UDCs starting with the very beginning of the UDC directory (rather than just starting with the currently-executing UDC) — this lets a UDC call itself recursively (either directly or indirectly, e.g. when UDC A calls UDC B which in turn calls A).
- **NORECURSION**, which turns off **OPTION RECURSION** and allows UDCs to only call those UDCs which come after them in the UDC directory. (This is the default).

OPTION LIST can be useful when you're debugging a UDC and want to check which commands are being executed; however, the **MPEXCMDTRACE JCW** (documented elsewhere in this manual) is probably more useful for this purpose. **OPTION RECURSION** could be convenient in some cases, but you might just want to use command files (which can always be used recursively) instead.

%PAUSE

Syntax: `%PAUSE numseconds`

Examples: `%PAUSE 300`

Suspends MPEX for the given number of seconds.

%PAUSEJOB, %SHOWPAUSED

Syntax: `%PAUSEJOB "[{time|booleanexpression}]" , userset
[;VERIFY|NOVERIFY|QUIET]
[;scheduleparms]
[;CHECKEVERY=minutes]

%SHOWPAUSED`

Examples: `%PAUSEJOB "5:30PM", #J123
%PAUSEJOB "11:00PM", COMPILE,@.@; NOVERIFY
%PAUSEJOB "JSCOUNT(' @,REPORT.@&EXEC') <=3", REPORT,@.@
%PAUSEJOB , #J987; IN=0,0,120`

The **%PAUSEJOB** command allows you to **SUSPEND** (via the **:BREAKJOB** command) a particular job or jobset (using VESOFT usersets — see the usersets section of the SECURITY User Manual for details), and have MPEX automatically **RESUME** the job at a specified future time or when a given expression becomes true.

The syntax for %PAUSEJOB is quite similar to %ALARM (documented earlier in this manual). In fact, you must have the ALARM task running under the BACKG job in order for %PAUSEJOB to be able to RESUME the job later (%PAUSEJOB actually sets a special type of alarm to later :RESUMEJOB the suspended job). For this reason, if you issue a %PAUSEJOB command and the BACKG job is not already running, MPEX will attempt to start it for you. For more information on the BACKG facility, please see the documentation for the [%BACKG command](#), earlier in this manual.

You may tell %PAUSEJOB to RESUME the suspended job:

- at a particular future time (e.g. "5:30 PM");
- when a particular boolean expression becomes true (e.g. "JSCOUNT('COMPILE,@.@')<=3"); or
- using the MPE :STREAM-style scheduling parameters ;AT=, ;DAY=, ;DATE=, and ;IN= (e.g. ";IN=0,0,5" or ";DATE=11/05/91;AT=9:00").

If you specify a jobset (rather than a single job), %PAUSEJOB will list all of the jobs that will be affected and ask

```
OK to process these jobs (y/n)?
```

If you don't want to be asked, use the ;NOVERIFY keyword. If you don't even want %PAUSEJOB to list the jobs that it processes, use the ;QUIET keyword (which implies ;NOVERIFY).

Performance considerations (;CHECKEVERY=)

By default, the ALARM background task wakes up once per minute to see if any of the %PAUSEJOB conditions have been met (i.e., the "logical expression" returns TRUE). If you set a %PAUSEJOB alarm based on an expression that will use a lot of system resources (CPU, disk I/Os, etc.) to evaluate, you may want to check that particular alarm less frequently.

When you enter a %PAUSEJOB command, simply append the ;CHECKEVERY= keyword to tell the ALARM background task how many intervals to skip between tests. For example:

```
%PAUSEJOB "JSCOUNT('COMPILE,@.@&EXEC')=0", #J115; CHECKEVERY=10
```

This means ":BREAKJOB #J115, and when there are no more jobs executing that logged on with a jobname of COMPILE, :RESUMEJOB #J115." Since the default pause time is one minute, this will wait 10 minutes between tests. Please see the documentation on the [%ALARM](#) and [%BACKG](#) commands earlier in this manual for more details.

Viewing "paused" jobs (%SHOWPAUSED)

To find out what %PAUSEJOB commands still have :RESUMEJOB alarms pending for them, use the %SHOWPAUSED command. For each "paused" job that is still suspended, you will be shown the job number, the intro time, the "will resume" time (unless the %PAUSEJOB was set to resume based on a logical expression), and the job name.

%PRINT

Syntax:

```
%PRINT [fileset | (file-eq-parms)]
        [;BUF]
        [;COPY[ACCESS]]
        [;FORMAT= formatstring]
        [;HIGHLIGHT]
        [;KEEPAMDATES]
        [;NOFILEPAGEBREAK]
        [;NONUMRECOGNIZE] [;{UNN|NUM}]
        [;{OLD|OLDANY|OLDTEMP}]
        [;OUT= {outfile | (file-eq-parms)}]
        [;PAGE= pagesize] [;PAGEHEAD]
        [;PAGEHEAD]
        [;SEARCH= expression]
        [;ALLFILENAMES]
        [;CONTEXT= [{numlines1 | exp1}], [{numlines2 | exp2}]]
        [;MAX= numlines]
        [;PREV= expression]
        [;START= startrecnum] [;END=endrecnum]
        [;WAIT=[ [+ | - ] numseconds]]
```

Examples:

```
%FILE LP;DEV=LP;CCTL
%PRINT AP@.SOURCE(CODE="EDTCT");NUM;KEEPAMDATES &
        ;OUT=*LP;PAGEHEAD
%PRINT MYFILE;START=100;END=150;NUM
%PRINT PROGFILE;FORMAT=STRWRITE(R: 'GARBAGE')
%PRINT @.SOURCE;SEARCH="CUSTNO" or "CUSTNUM" or "CUSTNAME"
%PRINT @.SOURCE &
        ;SEARCH=("FOO" and "BAR") or ("XYZ" and NOT "ABC")
%PRINT @.DOC;SEARCH=CASELESS "DEBUG"
%PRINT @.SOURCE;SEARCH=DELIM "CUST"
%PRINT @.SOURCE;SEARCH="FNAME";PREV="PROCEDURE" or "FUNCTION"
%PRINT @.SOURCE;SEARCH=CL"PAR-ONE";CONTEXT="PERF","END-PERF"
```

The %PRINT command lets you conveniently output filesets (or individual files) to the terminal, to the printer or to some external file. Major features include:

- You can print entire filesets, e.g.

```
%FILE LP;DEV=LP
%PRINT @.SOURCE(CODE="EDTCT"); NUM; OUT=*LP; PAGEHEAD
```

This will print all the files in the SOURCE group that are EDITOR /SET FORMAT=COBOL files (CODE="EDTCT") with line numbers to the line printer.

This is more convenient than

```
%EDIT @.SOURCE (CODE="EDTCT"), LIST ALL,OFFLINE
```

because it doesn't print any of those pesky one-page /VERIFY ALLs that EDITOR is so fond of. (The %PRINT command is also faster.)

- When specifying the file to be printed or the ;OUT= file, you can specify not just a filename, but an entire file equation (except for the word ":FILE") — you can even use the special extensions described under VEFOPEN in Appendix). Thus you can say

```
%PRINT MYFILE;OUT=(LP;DEV=LP,5,2;ENV=MYENV.HPENV.SYS)
```

or

```
%PRINT (XFILE,OLD;EXC);SEARCH=...
```

Note that the "*filename*" **MUST** be enclosed within parenthesis. This indicates to MPEX that the keywords that follow (;DEV, ;REC, etc.) are for the :FILE to be processed or used as output and not as parameters to the command itself.

This essentially saves you from having to set (and :RESET) a special file equation; however, it is especially useful with our STDLIST function (see [Appendix B](#)). A quick example:

```
%PRINT (![STDLIST('RUN MYPROG')]); &
      SEARCH="ERROR"; CONTEXT=,2; &
      FORMAT=STRCHANGE(R,"ERROR","*****ERROR*****  ")
```

This runs the program MYPROG and passes its output to the %PRINT command, which prints all of the lines that contain the string "ERROR" (+ the 2 following lines), changing "ERROR" to "*****ERROR***** " for emphasis!

- When you're outputting to the terminal, the display will pause every 23 (or however many you specify) lines — when you press [RETURN], it will resume printing.
- You may (by specifying the ;NUM keyword) see the line number of each line output — this is either the absolute record number or, if the file is numbered, the true line number from the file.
- You can indicate (with ;START= and ;END=) which records of the file you want to output.

For instance, if you say

```
%PRINT MYFILE; START=-8; NUM
```

it will display — with line numbers — the last 8 lines of the file:

```
99          PROC (CAPS)
100         ELSE IF MINUS'CAPS <> 0D THEN
101.1        PROC (DNOT (MINUS'CAPS))
101.5        ELSE
102          PROC (-1D);
110         END;
111        END;
113       END.
```

If you had said

```
%PRINT MYFILE; START=10; END=60; NUM
```

it would have displayed the 10th through the 60th line of the file; however, since this would fill up more than one terminal screen, it would pause every 23rd line and say, for example:

```
31                                "or all - capabilities" TO' TERMINAL
32                                ELSE IF CAPS <> 0D THEN
(33/97) Continue (Yes/No/Quit/Dontask)?
```

At this point, you could:

- Press [RETURN] or type Y (for "Yes") to see the next 23 lines.
- Type N (for "No") to stop printing this file.
- Enter a positive integer (e.g. 50) to continue printing the file starting with record number 50 (counting from 1).
- Enter an integer prefixed by a + (e.g. +5) to continue printing the file starting with record number 38 (=33+5, where 33 is the number printed before the "/" in the "Continue?" line).
- Enter an integer prefixed by a - (e.g. -10) to continue printing the file starting with record number 23 (=33-10, where 33 is the number printed before the "/" in the "Continue?" line).
- Press Q (for "Quit") to stop printing all files in the current fileset.
- Press D (for "Dontask") to print the rest of this file and all of the files in the current fileset, without stopping for a "Continue?" prompt again.

(The first number in the parentheses — 33 — is the current record number; the second number — 97 — is the total number of records in the file.)

%PRINT keywords

- **;BUF** — used to avoid stack overflow errors. By default, the %PRINT command reads files NOBUF, which makes fileset searches more efficient.

Unfortunately, it also uses more stack space; it is possible (though unlikely) that if your **;SEARCH=...** and **;PREV=...** selection conditions are complicated enough, you will get a stack overflow.

If this happens, just specify **";BUF"** on your %PRINT command, e.g.

```
%PRINT @.SOURCE;SEARCH=...;BUF
```

This will tell MPEX to do buffered reads, which will use less stack space but will also be slower.

- **;COPY[ACCESS]** — uses non-destructive reads for message files. MPEX's %PRINT command uses ordinary intrinsics to read a file. In the case of a message file, records that have been read from the file are deleted — nothing special, this is exactly what is supposed to happen when reading a message file. However, when you are searching a fileset that contains a message file, reading (and therefore emptying) a message file may have disastrous side effects on your production environment. Another side effect of reading a

message file is that if the file is empty to begin with, the read will wait or "hang" until someone actually WRITES to the file. When this happens, MPEX appears to "hang" and you have to press [BREAK] and type :ABORT to exit the program.

When you use ;COPYACCESS (which may be abbreviated to just ;COPY), MPEX will use a special mode when reading message files that preserves the contents of the file and will not "hang" if the file is empty.

- ;FORMAT= lets you tell %PRINT not to output the actual record from the file, but rather to output some function of the record. Here are some examples and their explanations:

```
%PRINT...;FORMAT=R[0:10]
```

Displays just the first 10 characters of each record.

```
%PRINT...;FORMAT=STRWRITE(R: 'GARBAGE')
```

replaces unprintable characters with a ".".

```
%PRINT...;FORMAT=TOKEN(REMTOKEN(R, 'AN'), 'AN')
```

displays just the second "token" of each record (see the discussion of [TOKEN](#) and [REMTOKEN](#) in Appendix B for more details).

The ;FORMAT= parameter can be any expression whose result is a string.

- ;HIGHLIGHT will display "-----Printing: *filename*" in inverse video. This is most useful when printing a fileset using ;SEARCH=" and displaying only a few lines from each file. When ;HIGHLIGHT is specified on the print command, the filename will be printed in inverse video (if you are using an HP terminal or emulator). This makes it easy to distinguish the filename from the text of each file being printed.
- ;KEEPAMDATES — indicates that the files' last Access and Modification dates should not be changed as the files are printed or searched; this is really useful if you want to print or search a whole fileset but still want all those files to be eligible for archival if they haven't been accessed in a long time.

If you say :SETJCW MPEXPRINTKEEPAMDATES=1 in your MPEXMGR.PUB.VESOFT file, the ;KEEPAMDATES keyword will always be in effect for the %PRINT command. (see also the discussion on [MPEXDEFAULT xxx variables](#)).

- ;NOFILEPAGEBREAK — tells MPEX not to do a page break after each file it prints. This is useful if you're doing a %PRINT...;SEARCH=... and are therefore printing only one or two lines from each file; you don't want to have a page break every one or two lines.
- ;NONUMRECOGNIZE — indicates that %PRINT should print the full text of the file despite the fact that the last 8 characters of the first line are digits. If you don't specify this keyword, %PRINT will think that the file is a numbered file, and treat the last 8 digits as the line number; this is usually the right thing to do, but for files that contain actual numeric data in the last 8 columns, ;NONUMRECOGNIZE is the way to go.
- ;UNN — indicates that output should not contain record numbers (whether or not the file has internal line numbers). This is ordinarily the default, but if you have set a VESOFTDEFAULTPRINT variable with ";NUM", then this will override the variable for this command only.

- **;NUM** — indicates that output should contain either line numbers (if the file is numbered) or record numbers (if the file is unnumbered). This is most useful when printing source files that are kept numbered (such as EDTCT files). See the section on **;SEARCH=** for another useful facet of this keyword.
- **;OLD** means "process permanent files only."
;OLDTEMP means "process temporary files only."
;OLDANY means "for each file in the fileset, first look for the file as a temporary file and then, if none exists, as a permanent file".

What is the default? Well, if you specify a single file (i.e. a filename with no wildcards, no indirect file, no + or -, and no selection condition), the default is **;OLDANY** (first TEMP, the PERM); if you specify a fileset, the default is **;OLD** (PERM files only).

Note: For compatibility with MPE/iX's **:PRINT** command, if you type **"%:PRINT filename"** or **"%PRINT filename"**, MPEX will look first for a temporary file and then, only if it doesn't find one, a permanent file. However, a **%PRINT** fileset command will NOT print any temporary files because this is a fileset-handling command and we don't want to surprise you by printing a whole bunch of permanent files and one temporary file that happens to have the same name as a permanent file.

- **;OUT=filename** — indicates that output is to go to the specified file, such as the line printer; for example, to output to the line printer, you might do a **:FILE LP;DEV=LP** and then a **%PRINT fileset;OUT=*LP**.

(Naturally, if this option is selected, you won't be prompted with **"(xx/yy) Continue?"** every so many lines.)

- **;OUT=(file-eq-parms)** — indicates that output is to go to the file or device specified in the file-eq-parms without using a **:FILE** equation. For example, **%PRINT fileset;OUT=(LP;DEV=LP)** will perform the same print as above (sending the output to the printer) without actually setting a file equation.
- **;PAGE=pagesize** — indicates how many lines to print before prompting with **"Continue (Yes/No/Quit/Dontask)?"**.

The default is 23 when outputting to the terminal, 60 when outputting to the printer (no prompt in this case).

- **;PAGEHEAD** — indicates that page headers (including the name of the file, the date and time, and the page number) are to be output to the **;OUT=** file. Useful when the **;OUT=** file is the line printer. If you use this option to print to the line printer, you should specify **;CCTL** on your file equation for your **;OUT=** file (**:FILE LP;DEV=LP;CCTL**).

Default is that no page headers are to be output — this is for compatibility with MPE/iX, although it's admittedly not very nice.

- **;SEARCH=searchstring** — restricts output to only those lines containing the search string (or search expression). This is perhaps the most powerful aspect of the print command and is documented in greater detail below.

The following **%PRINT** keywords are useful only when **;SEARCH=** has been specified:

- **;ALLFILENAMES** — display all filenames as files are searched. Ordinarily, when using **;SEARCH=**, MPEX will not display the name of each file being searched unless the file contains the search string.

Specifying `;ALLFILENAMES` causes MPEX to print each file as it is searched, regardless of whether or not the file contains the search string. This is useful if you want to watch the progress of the command on a very large fileset when only a few files contains the search string.

- `;PREV=prevsearchstring` — searches backwards for a previous occurrence of a different search string.

Let's say that you want to look through your programs for all occurrences of a particular variable. Chances are that you don't just want to see the lines that mention the variable but that you also want to know in which procedure or function (in PASCAL terminology) the variable is referenced.

For example, to see both the line that contains the word "CUSTNUM" and the last prior occurrence of the line that contains "PROCEDURE" or "FUNCTION", the following command will do exactly that:

```
%PRINT @.SOURCE;SEARCH="CUSTNUM";PREV="PROCEDURE" or "FUNCTION";NUM
-----Printing MYPROG.SOURCE
  11.5  PROCEDURE ERRORPROC (VAR F: INTEGER;
  15      WRITELN (CUSTNUM:10, ' NOT FOUND!');
  922    FUNCTION COPYREC (VAR FILE: TSTRING): BOOLEAN;
  939      IF CUSTNUM<>'XXX' THEN
  992.44  CUSTNUM:=' ';
  ...
```

As MPEX reads each record from the `SOURCE` group files, it checks if the record contains `PROCEDURE` or `FUNCTION`; if it does, MPEX doesn't print the record but rather saves it away. Then, when MPEX actually finds a record that refers to `CUSTNUM`, it prints both the saved `PROCEDURE/FUNCTION` record and the `CUSTNUM` record.

- `";CONTEXT=[startrec|startstring],[endrec|endstring]"` displays the line being searched for and the lines surrounding the search string. For example:

```
%PRINT MYFILE;SEARCH="CUSTNO" or "CUSTNUM";CONTEXT=-1,+2
```

will print 1 line before and 2 lines after each line that contains "CUSTNO" or "CUSTNUM".

But wait — there's more! What if you don't want to show a fixed number of lines before or after, but show the entire logical unit that contains the line found by `;SEARCH=`, no matter how large it is? Here's an example of how you might do this:

```
%PRINT MYFILE;SEARCH="CUSTNO" or "CUSTNUM";&
CONTEXT=("PROCEDURE" or "FUNCTION"), (R[0:3]="END")
```

Here, MPEX will find all the lines containing the string "CUSTNO" or "CUSTNUM", and then print all the lines from the immediately preceding line that contains "PROCEDURE" or "FUNCTION" and until the immediately following line that contains "END" as its first three characters. This is, of course, appropriate for PASCAL programs; however, other languages and files will have similar `;CONTEXT=` strings. For instance, to display all the UDCs in a particular file that `:RUN` a particular program, you can say

```
%PRINT UDC1;SEARCH="RUN PRG";CONTEXT=(R[0:1]="*"), (R[0:1]="*")
```

The `;CONTEXT=` in this example indicates that all lines surrounding the `RUN` command, from the preceding line with an "*" in column 1 to the following line with an "*" in column 1 are to be printed.

The general syntax of the `;CONTEXT=` keyword is thus:

```
;CONTEXT=[-numlines1], [+numlines2]
           [exp1]           , [exp2]
```

The number or expression before the comma indicates which lines before the line containing the found string are printed; the number or expression after the comma indicates which lines after that line are printed.

As you may have noticed, we've used both strings (e.g. `;CONTEXT="PROCEDURE"`) and logical expressions (e.g. `;CONTEXT=(R[0:1]="*")`) in our examples. This corresponds to the way `;SEARCH=...` and `;PREV=` work — they can take either strings (which mean "search for this string") or expressions (which mean "search for lines for which this expression is true"). A string in a `;CONTEXT=` indicates "print lines up to and including the one that contains this string"; an expression indicates "print lines up to and including the one for which this expression is true".

- `";MAX=numlines"` — indicates the maximum number of lines to search and print. This is also used to optimize the `FSEARCHEXP(...)` function when searching for a string in a fileset. Ordinarily, the `FSEARCHEXP(...)` and similar functions read the ENTIRE file looking for the string specified. On especially large files, this can take quite some time, but if all you are interested in is whether or not the file contains the string at all, stopping the search after you have found the first occurrence will save the time it takes to search the rest of the file.

For instance, if you wanted to `%COPY` all of the files that contain a reference to a particular file (perhaps in preparation for changing the structure of that file), the following command could be used:

```
%COPY @.FROZEN(FSEARCHEXP("'CUSTFILE';MAX=1")>0),=.WORKING
```

If you use `;MAX=numlines` with `;CONTEXT=range`, the `%PRINT` command will display the entire range for each occurrence of the `;SEARCH=` string.

- `";START=startrecnum"` — indicates the starting record number for the output. Positive numbers indicate record numbers from the start of the file (1 being the first record); negative numbers indicate record numbers from the end of the file (-1 being the last). Default is 1 (the first record).
- `";END=endrecnum"` — indicates the ending record number for the output. Positive means from start of file (1 = first record); negative means from end of file (-1 = last record). Default is -1 (the last record).
- `";WAIT=[numseconds]"` — indicates that the file may be open for output by another process and that MPEX should "wait" when reaching the end of file or more data to be produced. If a number is specified, then MPEX will pause for the number of seconds specified. After this delay, MPEX will check to see if any additional data has been written to the file, and if so, continue printing until the end of the page or until all of the new data

has been printed. This is primarily useful when using the %PRINTO command to watch the progress of a job stream.

If the number of seconds specified is positive, MPEX will "beep" the terminal prior to resuming its printing. If the number is negative, MPEX will remain silent when displaying any new data.

The default is -5 (a five second delay without beeps).

One thing to be aware of is that when using ;WAIT=, the default pagesize (;PAGE=) will be set to 0 and the output of the print command will continue until the current EOF is reached.

Searching for strings (;SEARCH=)

The MPEX %PRINT command can not only print entire filesets, but can also search an entire fileset for a string. For instance, you can say

```
%PRINT AP@.SOURCE;SEARCH="CUSTNUM";NUM
```

and MPEX will print for you (with line numbers, as per ;NUM) all the lines that contain "CUSTNUM" in all the files in AP@.SOURCE. This can be quite handy in many ways — whenever you want to find all occurrences of a variable in a fileset of sources, of a filename in a set of job streams, of a word or phrase in a set of documents, and so on.

But wait — there's more! You can not only search for a single string but for MULTIPLE strings. For instance, if you say

```
%PRINT @.SOURCE;SEARCH="PROCEDURE" or "FUNCTION";NUM
```

MPEX will print for you all the lines that contain either the string "PROCEDURE" or the string "FUNCTION" (thus giving you an index of all the procedures or functions in your PASCAL programs). Similarly, you can say

```
%PRINT @.SOURCE;SEARCH="CUSTNUM" or "COMPNAME" or "DEFAULT";NUM
```

to print all lines that contain one of three strings, and so on.

Just like you can use OR to find all lines that contain one string OR another, you can use AND to find all lines that contain both strings:

```
%PRINT @.SOURCE;SEARCH="PROCEDURE" and "FNAME"
```

will find you all the lines that contain both the strings "PROCEDURE" and "FNAME" in the same line. Similarly, you can say

```
%PRINT @.JOB;SEARCH="!JOB" and NOT "MANAGER.SYS"
```

to find all the lines that contain "!JOB" but not "MANAGER.SYS".

Sometimes you want to search for a string regardless of whether it appears in UPPERCASE or lowercase. The CASELESS SEARCHING feature permits this. If you say

```
%PRINT @.DOC;SEARCH=CASELESS "INVOICING"
```

MPEX will find all occurrences of "INVOICING", "invoicing", "Invoicing" or any other UPPERCASE/lowercase combination. CASELESS can be abbreviated CL, e.g.

```
%PRINT @.DOC;SEARCH=CL "INVOICING" and "NO"
```

which finds all lines that contain "INVOICING" in either UPPERCASE or lowercase and at the same time contain "NO" in UPPERCASE. Note that the CL applies only to the string that immediately follows it (in this case "INVOICING") — "NO" will be matched only in UPPERCASE.

Say that you try searching for the string "WARD". If you just do a

```
%PRINT D@.DEV;SEARCH="WARD"
```

then you'll also find the strings "WARDEN", "FORWARD", "AWARDED", and so on. What if you just want to see occurrences of the string "WARD" when it occurs as a separate word, i.e. delimited by special characters?

You can do this by using the DELIM keyword (which can be abbreviated to just a D):

```
%PRINT D@.DEV;SEARCH=DELIM "WARD"      or
%PRINT D@.DEV;SEARCH=D "WARD"
```

This will find all the lines which contain the string "WARD" when the characters immediately before it and after it are non-alphanumeric characters.

What if you want to find all occurrences of a string delimited on one side by a special character? For instance, you might want to find all lines containing the string DB preceded (but not necessarily followed by) a delimiter – this might match all your database-related variables (i.e. DBTHISVAR, DBTHATVAR, and DB_THE_OTHER_VAR, but not BLOODBANK_VAR); you can do this by saying

```
%PRINT @.SOURCE;SEARCH=LDELIM "DB"
```

— the LDELIM means "delimited on the left"; similarly, RDELIM means "delimited on the right".

This can be especially useful for looking for all occurrences of a particular variable name in a source file (or a fileset of source files). However, since most languages allow at least one special character as part of a variable name ("_" in PASCAL or C, "-" in COBOL), you want to search for all occurrences of a string delimited by characters that are neither alphanumeric nor allowable in variable names.

To do this, we let you set a variable called MPEXPRINTNONDELIMS. If, for instance, you use COBOL, you can do a

```
%SETVAR MPEXPRINTNONDELIMS "-"
```

to tell MPEX that all %PRINT...;SEARCH=...DELIM (and LDELIM and RDELIM) operations are to consider "-"s as "non-delimiter" characters. The %SETVAR (like all %SETVARs) remains in effect for the remainder of your session (or until you do a %DELETEVAR MPEXPRINTNONDELIMS). If you want the non-delimiter characters to be always set to a particular value, you should put the SETVAR command into your MPEXMGR, MPEXMGR.PUB or MPEXMGR.PUB.VESOFT file

(documented elsewhere in this manual). You can specify more than one character in `MPEXPRTNNDDELIMS` — all the characters you specify will be viewed as non-delimiters.

Of course, you can combine `DELIM` (or `LDELIM` or `RDELIM`) and `CASELESS` to search for delimited strings ignoring case:

```
%PRINT AP@.SOURCE;SEARCH=DELIM CASELESS "FNUM" or
%PRINT AP@.SOURCE;SEARCH=D CL "FNUM"
```

The entire `;SEARCH=...`parameter is really just a logical expression (much like the logical expressions you can use in `:IF`, `:WHILE`, fileset selection conditions, etc.). One difference is, of course, that strings can be used in place of booleans — you normally wouldn't be able to say `"FOO" AND "BAR"`, but you can do this in the `;SEARCH=...`parameter.

However, you can actually specify an arbitrary logical expression as the `;SEARCH=...`parameter. This is especially useful because the current record being processed is available to you as a variable called `"R"`. Thus, you can, for instance, say

```
%PRINT @.DATA;SEARCH=(STR(R,10,5)="XYZZY");NUM
```

which will output all the records that contain the string `"XYZZY"` in characters 10 through 14 of string `R`. Similarly, you can say

```
%PRINT @.DATA;SEARCH=R MATCHES "@XY?#@";NUM
```

which will find all occurrences of the pattern `"XY?#"` in the `DATA` files. The `@` characters at the start and end of the string indicate that the pattern can be found anywhere in the string.

Just saying

```
%PRINT @.DATA;SEARCH=R MATCHES "FOO@";NUM
```

will find all the lines that start with `FOO`;

```
%PRINT @.DATA;SEARCH=R MATCHES "@BAR";NUM
```

will find all the lines that end with `BAR`.

Do you want to get really fancy? Try saying:

```
%PRINT @.DATA;SEARCH=VALIDINTEGER(STRRTRIM(STR(R,20,5)));NUM
```

This will find all the lines in all the files in the `DATA` group that have a valid integer in columns 20 through 24.

In addition to the variable `R` (the contents of the current record), the `;SEARCH=...`(or `;PREV=`) parameter can also refer to the variable `RECNUM` (record number) or `LINENUM` (line number). `RECNUM` refers to the record number of the current record in the file, starting with 1. Thus,

```
%PRINT MYFILE;SEARCH=(RECNUM MOD 2)=0
```

will output every other record in the file. (This can be particularly useful for reformatting files, especially if you use the `;OUT=` parameter to send the output to another file.)

LINENUM refers to the line number of the current record in the file. If the file is unnumbered, **LINENUM** is set to -1; if the file is numbered, **LINENUM** is set to the EDITOR line number MULTIPLIED BY 1000 — thus EDITOR line 1.5 has **LINENUM** = 1500.

You can use **LINENUM** to select a range of file records, not by record number (as **;START=** and **;END=** do) but by line number. Thus,

```
%PRINT AP@.SOURCE;SEARCH=BETWEEN(LINENUM,1000,9900)
```

will print lines 1.0 through 9.9 of all the files in **AP@.SOURCE**. Of course, you can do some even trickier things, for instance

```
%PRINT AP@.SOURCE;SEARCH=(LINENUM MOD 1000)<>0
```

which will find you all the non-integer line numbers in **AP@.SOURCE**. (Perhaps this might give you a good idea of changes that were made since the last renumbering.)

Why do we set **LINENUM** to an integer value (e.g. 1500 or 33250) instead of a real number (e.g. 1.5 or 33.25)? We would have liked to use real numbers, but unfortunately the HP3000's single-precision real numbers are not precise enough to correctly represent an 8-digit value; for large line numbers the error would be substantial enough to cause some serious problems.

If you're interested, the **%PRINT** command sets a variable called **MPEXPRINTLINESFOUND** to the number of lines that actually contain the given search string. (If no **;SEARCH=...**parameter is specified, **MPEXPRINTLINESFOUND** is set to the total number of lines in the files printed.)

A few final notes on using **;SEARCH=...**on the **%PRINT** command:

- When you use **;SEARCH=**, **CONTEXT=**, and **;NUM**, the lines that actually contain the string being searched for are marked with a "*" between the line number and the text; any lines shown as a result of a **;PREV=** are marked with a "-".
- If you omit the part of the **;CONTEXT=** parameter that comes before a comma, no lines before the one found are printed; if you omit the part that comes after the comma, no lines after the one found are printed.
- When you specify the "before" part of the **;CONTEXT=**, **%PRINT** has to read backwards from the point where it finds a line that matches the **;SEARCH=**. This is rather fast for fixed-record-length files, but much slower for variable-record-length and QEDIT files — keep this in mind.

String searches in fileset selections

You can use all the power of string searching in fileset selection conditions (using **FSEARCHEXP** or **FSEARCHSTRING**, which work like **%PRINT...;SEARCH=...**) — for instance,

```
%COBOL @.SOURCE(FSEARCHEXP("'CUSTNO' or 'CUSTNAME'")>0), =.PUB
```

will recompile all your **SOURCE** files that mention the variables **CUSTNO** or **CUSTNAME**! See the documentation on the file attribute functions **FSEARCHEXP** and **FSEARCHSTRING** in the MPEX Filesets chapter.

%PRINTI, %PRINTO

Syntax:

```
%PRINTI [ #Ispoolfilenumber ]      [ ;mpexprintparms ]
          [ #Jjobnumber             ]
          [   jobname               ]
          [ #Aschedjobnumber       ]

%PRINTO [{#0spoolfilenumber|#Jjobnumber|jobname}]
          [ ;mpexprintparms ]
          [ ;IFLOW ]
          [ ;SEARCH={IFTRUE|IFFALSE}]
```

Examples:

```
%PRINTI #J1234; SEARCH=":"
%PRINTO MYJOB,MYUSER.MYACCT; PAGE=9999; NUM
%PRINTO #01234; SEARCH="CIERR" or "FSERR"; CONTEXT=-2
%PRINTO ;WAIT;IFLOW
```

The %PRINTI and %PRINTO commands print, respectively, input and output spoolfiles; you may, optionally specify any of the parameters of the %PRINT command (see [above](#)).

If you specify a spoolfile number ("I1234" for %PRINTI, "01234" for %PRINTO), the given spoolfile will be printed.

If you specify a job number or job name, MPEX will find and print the \$STDIN (for %PRINTI) or \$STDLIST (for %PRINTO) spool file of the given job.

If you don't specify either a job number, job name or spoolfile number, the \$STDIN or \$STDLIST spoolfile of the last job submitted by MPEX (via %\$command, %SUBMIT or %SCHEDULE) or by STREAMX (see the SECURITY User Manual) or by VEAUDIT will be printed.

%PRINTI also works on all input spool files (which, however, exist only for currently running jobs, waiting jobs, and scheduled jobs, not jobs that have already run), but requires SM capability.

Using ;WAIT, ;IFLOW, & ;SEARCH=... to view \$STDLISTS

On MPE/iX, %PRINTO works on all output spool files, including ones currently being generated by a running job. There are a few %PRINT keywords that are especially well suited for printing \$STDLIST files. The ;WAIT= keyword has already been documented under the %PRINT command since it will work for any output file, not just \$STDLISTs.

When you supply the ;IFLOW keyword, %PRINTO will display the FALSE portions of :IF/:ELSE/:ENDIF blocks in low intensity. This makes it easier to scan the output of a \$STDLIST file since you can tell at a glance that a particular statement was or was not executed without having to look back (perhaps several screens) for the "*** CONDITION FALSE, COMMANDS IGNORED..." message buried in the \$STDLIST.

The ;SEARCH= attribute IFTRUE takes the above concept a step further and allows you to filter out the false portion of a job stream entirely. Or, if you prefer, you can use ;SEARCH=IFFALSE to just view the parts of a job that were not executed. In either case, if

something appears in the listing that should not have, then it is a simple matter to look back at the preceding condition to determine why it was not evaluated correctly.

%PROGINFO

Syntax:

```
%PROGINFO progrname
```

Examples:

```
%PROGINFO DBUTIL.PUB.SYS
```

Note: %PROGINFO only works on Compatibility Mode (CM) programs.

Ever want to find out some detail about a program file? Well, here's a command that will display a great deal of information about a compatibility mode program.

For example, %PROGINFO QUERY.PUB.VESOFT might show the following:

```
FILE: QUERY.PUB.VESOFT      CAPS   : BA IA MR      MAXDATA: 32767
DL   : 0                    STACK  : 1200          DBSIZE  : 12830
DB AREA RECNUM: 1           EXTERNALS RECNUM: 862    ENTRY RECNUM: 872
FPMAP RECNUM: 873
CREATED 91/03/25,  2:21 PM    LAST ACCESSED 91/06/28, 10:05 AM
SEG  LEN REC#    SEG  LEN REC#    SEG  LEN REC#    SEG  LEN REC#
  0  2260  102      1  2176  120      2  7516  137      3  6396  196
  4  7544  246      5  2072  305      6  8456  322      7  8124  389
...

ENTRIES: QUERY'C'00'00(main) BUGS SEARCH
EXTERNALS: ASCII BIMAGEVERSION BINARY CALENDAR CATCLOSE CATOPEN CATREAD
           CAUSEBREAK CLOCK COMMAND COMPLIBINFO DASCII DBBEGIN DBCLOSE
           DBCONTROL DBDELETE DBEND DBERROR DBFIND DBGET DBINARY DBINFO DBLOCK
           DBMEMO DBPUT DBUNLOCK DBUPDATE DEBUG DFLOAT' DIVD DLSIZE EXTIN'
...
           SORTOUTPUT SORTTITLE TERMINATE TERMINATE' UNLOADPROC VEOPEN WHO
           XARITRAP XCONTRAP ZSIZE
```

As you can see, this copy of QUERY doesn't call **DBOPEN**! Instead, it calls **VEOPEN**, VESOFT's improved version of **DBOPEN** (see the section Database Security in the SECURITY User Manual for details).

%PURGE

Syntax:

```
%PURGE filesset [,TEMP]
                [;YES]
```

Examples:

```
%PURGE K#####.@.@+LOG####.PUB.SYS
%PURGE @,TEMP;YES
%PURGE @.@.@ - @.@.VESOFT(ACCDATE<TODAY-120 and NOT ISPRIV)
%PURGE /      - /VESOFT/ (ACCDATE<TODAY-120 and NOT ISPRIV)
```

The %PURGE command lets you — you guessed it — purge entire filesets, saving a lot of disk space in the process. For instance,

```
%PURGE K#####. @. @+LOG####. PUB. SYS
```

will purge all your EDITOR K-files and your system log files;

```
%PURGE @. @. @ - @. @. VESOFT (ACCDATE<TODAY-120 and NOT ISPRIV)
%PURGE /      - /VESOFT/ (ACCDATE<TODAY-120 and NOT ISPRIV)
```

will purge all the files in the system that haven't been accessed in 120 days except for databases or other PRIV files (You might do an %MPEXSTORE first to :STORE them to tape and then do the %PURGE.)

The ,TEMP option allows you to %PURGE temporary filesets (just like MPE's :PURGE ,TEMP allows you to :PURGE single temporary files)!

Another application for %PURGE is purging all of the files in an account:

- without locking up the directory (which the :PURGEACCT does, sometimes for minutes, while it purges all the files);
- without destroying the users and groups within the account.

If you want to save space by deleting an account (e.g. the contributed library, TELESUP, etc.), you can :STORE all the files and then do a

```
%PURGE @. @. TELESUP
```

This will delete the files but leave the accounting structure intact — whenever you want to :RESTORE any files into the account, you won't have to worry about specifying the ;CREATE keyword or making sure all the groups have the right capabilities and access masks, etc.

Also note that the %PURGE command allows a system manager, an account manager (in his own account) or a database's creator to %PURGE a database just like he would a normal fileset.

Automatic verification to avoid mistakes

A natural concern that most people have is: what if I accidentally mess up? You meant to say %PURGE @. @. @-@. @. VESOFT (ACCDATE<TODAY-120), but instead you typed:

```
%PURGE @. @. @ - @. @. VESOFT (ACCDATE>TODAY-120)
```

Wow! Now that wouldn't be very nice.

As the old saying goes, "to err is human, but to really foul things up, you need a computer". Just as the power of MPEX can make it very easy to do things that you want to do, it can make it very easy to accidentally do things that you don't really want to do.

Fortunately, MPEX has some very important fail-safes that can vastly decrease your chances of error. The most important one is that:

- **Before doing ANY real %PURGEs, MPEX will first show you the names of all the files that are to be purged.**

MPEX COMMAND REFERENCE: %PURGE

For instance, let's say you type:

```
%PURGE SA@.SOURCE
```

The output you get might look like this:

```
-----Will purge SAMSFILE.SOURCE.AP (code "EDTCT", type FA, 1333 sectors)
-----Will purge SAR010X.SOURCE.AP (code "EDTCT", type FA, 4001 sectors)
-----Will purge SATURDAY.SOURCE.AP (code "", type FB, 128 sectors)
-----Will purge SAZ999.SOURCE.AP (code "", type VAM, 22 sectors)
5484 sectors will be saved
OK to purge 4 files (y/n)?
```

So far, none of the files have actually been touched. If you say "N" (or press [RETURN]), MPEX will say

```
Error: Operation aborted by user request.
```

and the files will remain intact.

If, however, you answer "Y" to the "OK to purge?", MPEX will actually purge the files:

```
-----Purging SAMSFILE.SOURCE.AP (1333 sectors)
-----Purging SAR010X.SOURCE.AP (4001 sectors)
-----Purging SATURDAY.SOURCE.AP (128 sectors)
-----Purging SAZ999.SOURCE.AP (22 sectors)
5484 sectors saved
```

This way, if you inadvertently specify the wrong fileset, you'll get to see all the files that are to be purged before they're actually touched. Then, if you see your production database come up on the list, you'll know that something's wrong...

Note that when MPEX shows you the "-----Will purge ..." information, it not only shows you the filename, but also the filecode, the record type (which is the same as the "TYP" column in a :LISTF...,2 listing — FA, for instance, means Fixed Ascii), the file size in sectors, and the number of records in the file (not shown in the above example). This gives you an additional chance to make sure that the files being purged are actually the ones you want to purge.

A few other alternatives exist if you want to be really secure about all this. For instance, you can add to your MPEXMGR.PUB.VESOFT file (which is automatically executed whenever anyone enters MPEX) the commands:

```
%SET CAPABILITY,PURGE,AM
%SET DEFAULT,PURGE,?
```

- The first command (%SET CAPABILITY) indicates that the %PURGE command can only be used by people with AM capability working inside their own accounts (except that SM users can always use any command).
- The second command says that if a user types "%PURGE" (rather than "%!PURGE" or "%\$PURGE"), he'll be prompted Yes/No for each file in the fileset, as well as once for the entire fileset.

If you're really concerned about accidental damage done with the %PURGE command, you may use one or both of these commands; however, we believe that the fileset-level YES/NO verification that the %PURGE command now provides should be more than enough.

If, on the other hand, you do not want to have all this verification — if you want the %PURGE to be done without prompting — you can add ";YES" to the command.

As an alternative, you can set the JCW MPEXYESPURGE to 1 to cause all %PURGE commands to act like %PURGE...;YES. We implemented this because a number of users requested it; we do not recommend that you do this, but if you want to live dangerously, go ahead.

A better alternative is to set the MPEXYESPURGEONE JCW to 1. This means "don't ask yes-or-no when the %PURGE command specifies a single file rather than a fileset." This is especially useful when you want to execute an MPE/iX command file and don't want to be prompted every time it attempts to purge a single file.

Note that if you execute a %PURGE command in batch (by saying %\$PURGE..., %SCHEDULE PURGE... or %SUBMIT PURGE...), the %PURGE command will (naturally) not ask for verification, since it'll be running in batch where there's nobody to ask.

However, remember that whenever you submit a job via \$, %SCHEDULE, or %SUBMIT, MPEX asks you "OK to stream (y/N)? " — this is your last chance to say "whoops! I didn't really mean it!".

%PURGELINK

Syntax:

```
%PURGELINK fileset [;YES]
```

Examples:

```
%PURGELINK MYLINK
%PURGELINK ./LISTS/linkfiles
%PURGELINK ./Testdir/ ;YES
```

This command lets you PURGE symbolic link files created by the %NEWLINK command, as well as normal files and directories. Since symbolic link files are a POSIX related feature of the operating system, this command is only available on machines that support POSIX. (See the %NEWLINK command for a discussion of LINK files)

Note that like our %PURGE command for filesets, this command prompts for verification before actually purging the links and other files. If you are certain that the links can be purged without verification, then you can either use ;YES on the command, or set the variable VESOFTDEFAULTPURGELINK to ";YES".

It is important to realize that purging the LINK file does not purge the ACTUAL file, merely a pointer to the file. If you inadvertently purge a link file, the data is still in the original file.

Remember that, like MPE's :PURGELINK, this command purges ordinary MPE and POSIX files and directories as well as links. To purge only the link files in a fileset, type

```
%PURGELINK fileset(ISLINK)
```

%QEDIT

Syntax:

```
%QEDIT fileset, qeditcommand
```

Examples:

```
%QEDIT @.SRC, LIST "FCONTROL"
%QEDIT @.SRC, CHANGE "ACCT-NUM", "ACCOUNT-NUM", @
```

The %QEDIT command lets you execute QEDIT commands on an entire fileset. (QEDIT, a product of ROBELLE Consulting Ltd., is a powerful text editor that many of our customers — and we ourselves — use.)

For instance,

```
%QEDIT @.SRC, LIST "FCONTROL"
```

will display (to the terminal) all occurrences of the string "FCONTROL" in your SRC group files.

The command

```
%QEDIT @.SRC, CHANGE "ACCT-NUM", "ACCOUNT-NUM", @
```

will change all occurrences of "ACCT-NUM" to "ACCOUNT-NUM" in your SRC group files.

How it works

MPEX does slightly different things depending on the type of the file and on the QEDIT command being executed:

- If the command is a /LIST, MPEX uses QEDIT's "list external file" feature to /LIST the file without /OPENing it or /TEXTing it. This is quite a bit faster and also doesn't change the file's last modify date.
- If the command is not a /LIST but the file being processed is a QEDIT file, MPEX will have QEDIT /OPEN the file and execute the command.
- If the command is not a /LIST and the file being processed is not a QEDIT file, MPEX will have QEDIT /TEXT the file, execute the command, and /KEEP the file.

One thing you should be warned about is this: if you do a non-/LIST on a non-QEDIT file, MPEX will always tell QEDIT to do the /TEXT and /KEEP (but we do use QEDIT's "IFDIRTY" option). This is usually not a problem at all, since a useless /KEEP will just recreate the file.

However, in a few cases, it is possible that the rebuilding of the file by /KEEP might damage the file structure in some way — for instance, if the file is a KSAM file, a VFORM file or something like that, you might want to exclude it from the command.

You can easily do this with MPEX filesets:

```
%QEDIT @.SRC(ISQEDIT or ISASCII),C "ACCT-NUM","ACCOUNT-NUM",@
```

This will do the /CHANGE on only those files in the SRC group that have code QEDIT or are ASCII.

%QUIT

Syntax: `%QUIT [[CIERR=] errnum]`

Examples:

```
%QUIT
End Run

%QUIT 10
Program terminated in an error state. (CIERR 976)
```

The %QUIT command is a modified version of MPEX's %EXIT command. Unlike %EXIT, %QUIT always terminates the active MPEX process, without attempting to suspend MPEX inside of the current father process.

Practically, the only time you need %QUIT is when you're running MPEX from within a process-handling environment that isn't smart enough to allow you to re-activate suspended son processes (for example, when running copies of CI.PUB.SYS on an MPE/iX system).

%QUIT checks to see if MPEX has any active son processes (perhaps as a result of using the %GOON command), and will fail if any active son processes exist. You must either wait for the active sons to terminate or use the %KILL command to terminate them before %QUIT will succeed.

%QUIT also accepts an optional CIERR parameter. If a non-zero CIERR parameter is supplied, MPEX will terminate with the system JCW set to the value FATAL+CIERR. This allows you to signal to the father process that MPEX has terminated in an error state, and to pass an error code at the same time.

%REDO, %DO, %LISTREDO

Syntax:

```
%REDO      [ abscmdnum      ]  [, editstring]
            [ -relcmdnum      ]
            [ cmdnum1/cmdnum2 ]
            [ [""][@] string[""] ]

%DO         [ abscmdnum      ]  [, editstring]
            [ -relcmdnum      ]
            [ cmdnum1/cmdnum2 ]
            [ [""][@] string[""] ]

%LISTREDO  [ abscmdnum1[/abscmdnum2] ]  [ ;OUT=outfile]
            [ -relcmdnum[/-relcmdnum2] ]  [ ; {REL | ABS | UNN} ]
            [ [""][@] string[""]      ]
```

Examples:

```
%DO -2
%DO 155/158
%DO "MYUDC A,B"
%REDO 155
%REDO RUN
%REDO @MYFILE
%LISTREDO
%LISTREDO -30/-1

%,155          same as %REDO 155    (, abbreviates %REDO)
%,,           same as %LISTREDO    (,, abbreviates %LISTREDO)
%,.-6/-1      same as %DO -6/-1    (,. abbreviates %DO)
```

MPE/iX users might already be familiar with most features of these commands, since they're patterned on the corresponding MPE/iX commands.

Everybody makes mistakes (even you!). When you mistype an MPEX command, you'd like to be able to easily correct the typo without having to re-enter the whole thing. Say you entered

```
%RUN MYPROG;MAXDATA=30000;INFO="MYFILE.DATA.SYS";PARMN=555
```

You entered PARMN instead of PARM, and MPEX will give you a syntax error message.

Now, you can (just as in MPE) type

```
%REDO
```

and the command will appear for you to edit; after you make your modification (presumably deleting that "N"), you press [RETURN], and MPEX re-executes the modified command.

This is all well and good, but what if the bad command wasn't the last one you typed, but the second-to-last? You tried running the program, but it didn't exist; then you did a :LISTF to find out the true program name. Now the RUN command — all 57 characters of it — isn't the last command any more, but you still want to %REDO it.

MPEX lets you do this. By default, it saves in the command history the last 1000 commands you typed, and every one of them is available for you to

- Examine with the %LISTREDO command.
- Edit and re-execute via the %REDO command.
- Immediately re-execute (without editing) using the %DO command.

Because of its size, the command history can be used for much more than just correcting typos:

- You can have a sort of "instant function key" — say that you're working on a particular program and type a long %PREP command; next time you need to do the %PREP, just say

```
%DO PREP
```

and that command will be automatically re-executed. You only need to type the command once.

- You have a complete log of all MPEX commands typed in this MPEX session (we'll see later how you can save them permanently so they'll be accessible even after you exit MPEX and re-enter later). You can see whether you indeed did what you thought you did, whether the sequence of commands you typed which triggered a particular condition was what you thought it was, and so on.
- And, of course, you can still use %REDO to correct typos.

Referring to saved commands

In order to %REDO (or %DO) a command, you need to tell MPEX which command is to be redone. There are several ways of doing this:

- If you simply say

```
%REDO
```

MPEX will %REDO the last command you typed

- If you say

```
%REDO RUN          (%REDO followed by a string)
```

MPEX will %REDO the last command (if any) that starts with the string RUN.

- If you say

```
%REDO @MYFILE      (%REDO @string)
```

MPEX will %REDO the last command that CONTAINS the string MYFILE.

- If you say

```
%REDO 177
```

MPEX will %REDO the command with the ABSOLUTE COMMAND NUMBER 177.

How do you know a command's absolute command number? You might know it from a

```
%LISTREDO
...
176) PURGE MYFILE
177) PREP MYUSL,MYFILE;CAP=PH,PM,DS;MAXDATA=30000;RL=...
178) RUN MYFILE
...
```

or, if you've set MPEXPROMPT to include the command number (see [below](#)), you'll be able to see each command number on the screen as the command is entered:

```
(cmd176) % PURGE MYFILE
-----Purging MYFILE.DEV.VESOFT
(cmd177) % PREP MYUSL,MYFILE;CAP=PH,PM,DS;MAXDATA=30000;...
...
(cmd184) %
```

- If you say

```
%REDO -3
```

MPEX will redo the 3rd-to-last command you've typed. Your most recently typed command is referred to as -1, so

```
%REDO
```

is really the same as

```
%REDO -1
```

- If you say

```
%REDO 177/180
```

then MPEX will redo lines 177,178,179 and 180, prompting you for modifications for each and then re-executing each.

This is most useful on the %DO command (which re-executes a command without prompting for modifications): say that you've typed a sequence of 6 commands that you want to re-execute — you can type

```
%DO -6/-1      << relative or >>
%DO 179/184    << absolute >>
```

A good example of this is if you enter a multi-line >REPORT statement in QUERY (as we'll discuss later in the Running MPEX section, under "MPEX HOOK", VESoft makes the REDO facility available in QUERY, in EDITOR, etc.) — a single DO command can re-execute the entire report in one fell swoop.

Editing the command being %REDOne

In addition to letting you redo the last 1000 commands, MPEX makes it easier for you to edit them.

Instead of just the usual R, I and D editing characters, MPEX also lets you use:

>xxx	to append "xxx" to the end of the line.
>DDDDD	to delete five characters (one for each D) from the end of the line.
>Rxxxx	to replace the last 4 characters (in this case) of the line by "xxxx".
D>	to delete until the end of the line.
C/xx/yy	to change all occurrences of "xx" into "yy"; you may use any non-alphanumeric character like ' or " or / or , as a delimiter.

For example:

```
%REDO RUN
RUN myprod;LIB=G;MAXDATA=34000;INFO="TESTING 123"
D>
RUN myprod;LIB=G;MAXDATA=34000
>;PARM=555X
RUN myprod;LIB=G;MAXDATA=34000;PARM=555X
>D
RUN myprod;LIB=G;MAXDATA=34000;PARM=555
C/34/30/
RUN myprod;LIB=G;MAXDATA=30000;PARM=555
```

Try it out — it's a good deal more powerful than ordinary :REDO.

Actually, there's an even more powerful editing feature that you can use in :REDO (and also in EDITOR — see "[MPEX HOOK](#)" for more details). It is called VEMODIFY, and it uses the full power of your HP terminal to allow "full-line" modification. To use it, say

```
:SETJCW HPREDOVEMODIFY=1
```

See the [VEMODIFY](#) section for information on this. (Or try it — do the :SETJCW, :REDO a line, and press [Control-Q] to get help.)

Finally, if you know exactly what modifications you need to make, you can specify them on the %REDO or %DO commands:

```
%RUN MYFILE;LIB=G;STDIN=XFILE;INFO="BANANA";PRAM=123
Error: Expected one of the MPE :RUN parameters...
%DO ,C/PRAM/PARM/
```

Since we knew that the error was our "PRAM" typo, we immediately specified "C/PRAM/PARM/" to change the PRAM into PARM — this modification will be made and the command will be re-executed without any further prompting.

If you specify this edit string, %DO and %REDO behave identically: both execute the edit, and neither prompts for any more modifications.

Abbreviations for %REDO, %DO and %LISTREDO

Since %REDO is largely present to save you typing, it would be a shame if you had to say LISTREDO and REDO all the time — you might end up typing more than if you had just re-entered the entire command.

This is why we permit you to abbreviate all three commands:

,	in place of REDO
,,	in place of LISTREDO
,.	in place of DO

Thus, you might say

```
%,,
1079) FILE COPYLIB=MYCOPY
1080) $COBOL AP010S,AP010P,,,,,MAXDATA=30000
1081) $COBOL XFILE.SOURCE,XPROG.PUB,,,,,CAP=PH,DS;RL=MYRLFILE
...
%,.1079/1081
<< MPEX now re-executes commands 1079 through 1081 >>
FILE COPYLIB=MYCOPY
$COBOL AP010S,AP010P,,,,,MAXDATA=30000
OK to stream (Y/N)? Y
#J123
$COBOL XFILE.SOURCE,XPROG.PUB,,,,,CAP=PH,DS;RL=MYRLFILE
OK to stream (Y/N)? Y
#J124
<< MPEX is done re-executing 1079/1081; it prompts you again >>

%RUN MYFILE.DATA.PROD;LIB=G;INFO="BANANA";PARAM=177
PROGRAM FILE NOT FOUND. (CIERR 622)
%LISTF MYFILE.@.PROD,1
...
%,RUN
RUN MYFILE.DATA.PROD;LIB=G;INFO="BANANA";PARAM=177
C/DATA/PUB/
RUN MYFILE.PUB.PROD;LIB=G;INFO="BANANA";PARAM=177
...
```

As you see, in place of the LISTREDO, REDO, and DO commands, we used their abbreviations — in all other respects, the commands worked just as if they'd been fully spelled out.

%LISTREDO parameters

The %LISTREDO command has several parameters that are worth mentioning:

%LISTREDO	shows the last 20 commands you've typed.
%LISTREDO 10/50	shows commands #10 through #50 (pausing every 23 lines for you to press [RETURN]).
%LISTREDO 10	shows commands #10 through the most recent one.
%LISTREDO -25/-10	— you can use relative command numbers, too.
%LISTREDO RUN	shows all the commands that start with RUN.
%LISTREDO @MYFILE	shows all the commands that contain MYFILE.
%LISTREDO 10/15;ABS	shows commands with absolute command numbers (the default).
%LISTREDO 10/15;REL	shows commands with relative command numbers.
%LISTREDO 10/15;UNN	shows commands without command numbers.

%LISTREDO;OUT=*LP	sends all the output to the given file (*LP).
%LISTREDO 1	shows all the commands saved in the history (not just the last 20).

All of the above options — the command specification (1, 10/15, RUN, etc.), the numbering scheme (;ABS, ;REL or ;UNN) and the ;OUT= file may be combined any way you like.

Saving your command history as a permanent file

By default, your command history is kept in a program-temporary file. When you leave MPEX, all the saved commands are forgotten.

However, if you want to, you can save the command history so that even if you :BYE off and then log on again tomorrow, when you re-enter MPEX all the saved commands will be available to you.

To do this, just say

```
:FILE MPEX.REDO.VESOFT=MYREDOFL
```

MYREDOFL can be any file you want to; the first time you run MPEX with this file equation, MPEX will build MYREDOFL with all the right parameters. Then, every time you run MPEX after that (assuming the file equation is still in effect), MPEX will use that file as the "command history file"; any new commands you enter will be appended to the file.

As you can see, this is geared towards letting everybody have their own REDO files — you might even have a system logon UDC that includes the command

```
:FILE MPEX.REDO.VESOFT=MPEXREDO
```

This way (assuming everybody has his own group), each person will have his own command history file.

You can't have several people sharing the same command history file — MPEX opens it exclusively.

Setting MPEX prompt to make %REDOing easier

The one difficulty that you may sometimes face is identifying the command that you want to %DO or %REDO. There may be several commands that start with the same characters (so just saying %REDO RUN [or whatever] won't work); you could always do a %LISTREDO and try to find it there, but that might be more trouble than it's worth.

Many people like to address this by changing the MPEX prompt to include the MPEX variable HPCMDNUM. HPCMDNUM contains the command number of the current command — exactly the thing you need to later %REDO it. Thus, you might say

```
%SETVAR MPEXPROMPT "(!!HPCMDNUM) Yes, boss? "
```

MPEX will then prompt you with something like

```
(391) Yes, boss?
```

Now, if you ever want to %REDO or %DO the command that you enter here, you'll know that it is command number 391. As long as it's still on the screen (or in your terminal memory), the command number will be right there in front of you.

This is only one of the many nice things you can do with the MPEXPROMPT variable. You might, for instance, say

```
%SETVAR MPEXPROMPT "(!!HPCMDNUM !!HPUSER !!HPTIMEF) "
```

This will make every MPEX prompt show the command number, your logon user name, and the current time.

:REDO, :DO and :LISTREDO in other programs

As useful as :REDO, :DO and :LISTREDO are in MPEX, they can be even more useful in other programs, like EDITOR, QUERY, FCOPY, etc. We don't mean just :REDOing MPEX commands that you've executed from the program, but rather :REDOing the program's own commands.

For instance, you run :FCOPY, type a long FROM=;TO=;... command, and find that you've made an error — you'd like to use :REDO to redo that FCOPY command. You may type an eight-line >REPORT command in QUERY, and then find that you need to do another >FIND and :DO all the commands again.

The "MPEX HOOK" facility allows you to do exactly this, and more:

- It lets you :REDO, :DO and :LISTREDO all commands you type in a particular utility (with all of the features we talked about above).
- It lets you execute MPEX commands — including the compile commands, %RUN and UDCs — from within EDITOR, QUERY, etc.
- It lets you (if you have SM) save files in other accounts — /KEEP into other accounts in EDITOR, copy into other accounts in FCOPY, etc.

%REDO facility options

There are several options you can set for the REDO facility (which apply equally to the REDO facility available from MPEX HOOKed programs). They are set by defining various specially-named JCWs and/or variables. These JCWs must be set before issuing a %INITREDO command or running a HOOKed program within which you wish them to take effect.

:SETJCW MPEXREDOSIZE=200	Sets the number of commands saved in the history (default 1000).
:SETVAR MPEXREDOSIZE=200	On MPE/iX systems, you should use :SETVAR instead of :SETJCW. Note: due to the way MPE/iX handles the HPREDOSIZE variable, setting HPREDOSIZE outside of MPEX has no effect. Setting HPREDOSIZE inside MPEX is translated to MPEXREDOSIZE for compatibility.

:SETJCW HPREDOVEMODIFY=1	Turns on VEMODIFY mode in :REDO (default 0 = OFF).
:SETJCW VESOFTVEMODIFYBIT8CHECK=	(used for terminals that support 8-bit ASCII, such as a 2392, when using the EXTEND key) 0 if bit 8 is set, discard the character and beep 1 strip bit 8 2 accept full 8-bit characters
:SETJCW VESOFTVEMODIFYEXPANDOK=1	(used on 700/9x terminals and emulators) This JCW causes MPEX to use escape sequences to position the cursor which are not dependent upon the terminal width. By default, VEMODIFY assumes that your terminal is 80 columns wide, and uses escape sequences that won't work correctly if you have a 132 column wide terminal. VEMODIFY uses much longer escape sequences when you configure it this way, so we suggest you only use this mode when connected over a high speed line.
:SETJCW HPREDODEFLIST=200	Sets the default number of lines listed by a simple parameter-less %LISTREDO (default 20).
:SETJCW HPREDONODUPS=1	Tells MPEX that if exactly the same command is entered several times in a row, only one copy of it will be saved. Default (0): every command you type will be saved, even if it's the same as the last one.

Actually, we think that our defaults are pretty reasonable, but we give you these options just for additional flexibility.

%RELEASE

Syntax:

```
%RELEASE fileset
           [;KEEPAMDATES]
```

Examples:

```
%RELEASE MYDB+MYDB?#
```

Warning: The Programmer General has determined that **:RELEASE**ing files (in MPE or MPEX) can be dangerous to the health of your system security — see the [%SECURE command](#) documentation for more information.

MPEX COMMAND REFERENCE: %REMOTE HELLO

The %RELEASE command is just like MPE's :RELEASE, except that:

- It works on FILESETS.
- It does not require creator access. It can be used by anybody with Read, Write, and eXecute access to a file (since anyone who has Read/Write/Execute access can already do anything to the file).
- The ;KEEPAMDATES keyword allows you to release a file (or fileset) without changing that file's last Access and Modify dates.

Practically speaking, we don't encourage you to :RELEASE any files, much less filesets.

IMAGE databases are an exception — since they're protected by IMAGE security (and can't be accessed normally because of their "PRIV" filecodes), it's quite safe and often quite necessary to release them.

You can release a database very easily by saying:

```
%RELEASE MYDB+MYDB?#
```

(to :RELEASE the root file and all the datasets).

You can release all the databases in a group by saying:

```
%RELEASE @.DATA(ISPRIV)
```

Of course, you can %RELEASE any filesets you'd like — you might say

```
%RELEASE @.SOURCE(ISASCII)
```

to release your source files or what have you.

However, we advise you to :RELEASE files as little as possible, since :RELEASEing a file removes all security restrictions from it, allowing anybody to do anything to the file.

%REMOTE HELLO

Syntax:

```
%REMOTE HELLO remotelogon
```

Examples:

```
%REMOTE HELLO USER.ACCT;DSLINESYSB
%REMOTE:SYSB HELLO USER.ACCT
%DSLINESYSB
%REMOTE HELLO USER.ACCT
```

SECURITY's STREAMX and logon menus allow you to avoid having remote passwords embedded in job streams or menus; SECURITY will automatically prompt the user for the remote passwords or even insert them without prompting (if you configure the system for this). Of course, if you choose to have the passwords automatically inserted, you can control for which users and which job streams/menu files this will be done.

If you use SECURITY, you will also be allowed to do the same inside MPEX command files or UDCs. If, for instance, you want to have an MPEX command file that logs on to a remote

system, does some work on it, and immediately logs off, you can configure SECURITY to tell MPEX to insert the appropriate passwords into the REMOTE HELLO command (but only for this particular command file and only for a particular set of users that you can configure).

For more information on this, see "REMOTE HELLO Processing In LOGON MENU's And In MPEX" in the "STREAMX And !REMOTE HELLO" chapter of the BATCH ACCESS SECURITY section of the SECURITY User Manual.

Since this is fundamentally a SECURITY feature (implemented in MPEX only for additional convenience), it is available only for those MPEX customers who are also licensed to use SECURITY.

%RENAME

Syntax:

```
%RENAME fromfileset, tofileset
        [;KEEPAMDATES]
        [;CREATE[Q]]
        [;LOCAL[GROUPID]]
        [;DELACD]
```

Examples:

```
%RENAME MYDB@(ISPRIV), NEWDB@
%RENAME @.UTIL(ISPROG and PROG.PMCAP),@.PRIVUTIL
%RENAME @.OLDGROUP,@/= .NEWGROUP; KEEPAMDATES

%RENAME ./src/project11/@, ./arc/proj11/@ ;CREATE
```

The %RENAME command is quite similar to MPE's :RENAME, but with the following extensions:

- Of course, it works on filesets, including all the MPEX extensions (+ filesets, - filesets, and selection criteria).
- It works on IMAGE databases as well as normal files -

```
%RENAME MYDB@(ISPRIV), NEWDB@
```

will rename the database MYDB (the root file and all the datasets) into NEWDB. Note the ISPRIV selection criterion, which ensures that only the database (and no other files that may start with MYDB) is renamed.

- It does not require the user to be the file's creator. It can be used by anybody with Read, Write, and eXecute access to a file (since anyone who has Read/Write/Execute access can already do anything to the file – for instance, copy it and purge the original).

If you have SM capability, this command allows renaming across account boundaries!

- Note that MPEX's %RENAME, just like MPE's :RENAME, does not work on compatibility mode KSAM files. In fact, if you %RENAME either a KSAM data file or a KSAM key file, your file will probably become inaccessible until you %RENAME it back to its original name (again, this is just like MPE's :RENAME).

The only time when %RENAME will work on a KSAM file is if you rename both the data file and the key file into files with the same names as before but in another group. In this case

(e.g. when you do a %RENAME @.OLDGROUP,@.NEWGROUP), the files will be handled properly.

MPE/iX's Native Mode KSAM stores both the data and keys in a single file, which can be %RENAMED!

- The ;KEEPAMDATES keyword allows you to rename a file (or fileset) without changing that file's last Access and Modify dates.
- On systems that support POSIX, the ;CREATE keyword will create any directories needed to build the file. The ;CREATEQ keyword builds the directories without displaying any messages ("quiet" mode).

For example, to move all of the files in a source directory into an archive directory, building any necessary subdirectories along the way, you could use the last command in the above examples.

```
%RENAME ./source/proj11//@, ./archive/proj11//= ;CREATE

-----Renaming ./source/proj11/main.c  => ./archive/proj11/main.c
Directory  "/XYZCORP/DEV/archive/proj11" created
-----Renaming ./source/proj11/subs.c   => ./archive/proj11/subs.c
...
```

- Also for POSIX systems, there are additional MPE security restrictions placed on files that are not in their own "filegroup" (see the [%ALTFILE POSIX note](#) regarding file security for additional details). MPE uses ACDs and group/owner information to implement this additional security, and MPEX adds the same ACDs when renaming files from one account to another. MPEX has two options for %RENAME that allow you to %RENAME files from one account to another and have it actually belong to the new account:
 - The ;LOCALGROUPLD keyword (which may be abbreviated to just ;LOCAL) tells MPEX to rewrite the GROUPLD and OWNER information so that the file is owned by the account manager of the target account.
 - The ;DELACD keyword removes the additional ACDs that MPE applies when RENAMEing a file from one account to another. These ACDs enable the original owner or creator to maintain access to the file as if it were in the original account.

Some useful applications:

- Renaming databases, shown above.
- Moving files that match a particular selection criterion into a particular group. For example

```
%RENAME @.UTIL(ISPROG and PROG.PMCAP),@.PRIVUTIL
```

will rename all the privileged program files (PROG.PMCAP) from the UTIL group into the PRIVUTIL group (which might have more stringent security);

```
%RENAME @.DEV-S@.@(ACCDATE<TODAY-120),@.ARCHIVE
```

will move all the DEV files (except for those starting with S) that haven't been accessed in 120 days into the ARCHIVE group.

Lockwords and %RENAME

As we discuss in the MPEX And Your System Security section, MPEX is quite intelligent in its lockword handling. If you have SM capability (or AM capability and are working on files in your own account), MPEX will automatically supply the lockword for each file being processed.

This lets you say something like:

```
%RENAME @.OLDGROUP,@.NEWGROUP
```

without having to specify the lockword of every lockworded file in OLDGROUP.

However, if you execute this %RENAME command, what happens to all the lockwords of the files being renamed? Remember that we must stay compatible with the MPE :RENAME command.

If you say (in MPE)

```
:RENAME MYFILE/SECRET.OLDGROUP,MYFILE.NEWGROUP
```

then MYFILE will be renamed from OLDGROUP into NEWGROUP and its lockword will be removed. Don't blame us — this is how MPE does it; if the target filename does not include a lockword, the new file will be unlockworded.

Therefore, if you say

```
%RENAME @.OLDGROUP, @.NEWGROUP
```

MPEX will interpret this to mean "rename the OLDGROUP files into NEWGROUP and strip their lockwords" (just like the MPE :RENAME command).

What if you want to keep the lockwords? No problem! Just say

```
%RENAME @.OLDGROUP, @/= .NEWGROUP
```

The "=" in the lockword field means "keep the same lockword as the old file" (see "[Target Filesets](#)" in the MPEX Filesets chapter for more information on "=").

Therefore, when MPEX issues the :RENAME command for each file in @.OLDGROUP it'll be certain to include the old file's lockword (if any) on the new filename. Instead of saying

```
:RENAME MYFILE/SECRET.OLDGROUP,MYFILE.NEWGROUP
```

it'll say

```
:RENAME MYFILE/SECRET.OLDGROUP,MYFILE/SECRET.NEWGROUP
```

and all the lockwords will be preserved.

%REPEAT...%FOREACH

Syntax:

```
%REPEAT
%> command1
%> command2
...
%> command
%>FOREACH varname=value [,...]
```

Examples:

```
%REPEAT
%>ECHO Creating group and user for !NAME
%>NEWGROUP !NAME
%>NEWUSER !NAME; HOME=!NAME
%FOREACH NAME=BILL,GEORGE,SUE
```

The %REPEAT...%FOREACH construct performs a sequence of MPEX commands several times with a given variable (for example, "GROUP") set to each one of a given list of values:

```
%REPEAT
%> COPY MYFILE, MYFILE.!GROUP
%>FOREACH GROUP=PUB,DATA,JOB,LIB
```

will do the COPY command four times, setting the variable GROUP to the strings "PUB", "DATA", "JOB", and "LIB".

The items on the list can be delimited by commas or spaces; if an item has an embedded comma or space, enclose it in quotes.

%REPEAT...%FORFILES

Syntax:

```
%REPEAT
%> command1
%> command2
...
%> command
%>FORFILES filesset[:{SPOOL|TEMP}]
```

Examples:

```
%REPEAT
%>ECHO -----Processing file !MPEXCURRENTFILE
%>FILE MYFILE=!MPEXCURRENTFILE
%>RUN MYPROG
%>FORFILES @.DATA-T@.DATA(NOT ISPRIV)
```

The %REPEAT...%FORFILES construct allows you to execute an arbitrary sequence of commands on a filesset. When you type %REPEAT you will be prompted with "%>".

All commands that you type at the "%>" prompts will not be executed as you type them; instead, they'll be saved up until you type a command such as

```
%>FORFILES @.DATA-T@.DATA(NOT ISPRIV)
```

("FORFILES" followed by an MPEX fileset). Then, all the commands you input at the "%>" prompt will be executed once for each file in the specified MPEX fileset, with the variable MPEXCURRENTFILE being set in turn to the name of each file.

For instance, MPEX's fileset %PURGE command might very well be implemented as something like:

```
%REPEAT
%>:PURGE !MPEXCURRENTFILE
%>FORFILES fileset
```

For each file in the fileset, MPEX will execute the MPE command ":PURGE !MPEXCURRENTFILE".

If we wanted to do more, we could say

```
%REPEAT
%>ECHO -----Purging !MPEXCURRENTFILE
%>:PURGE !MPEXCURRENTFILE
%>FORFILES fileset
```

In reality, many MPEX commands are actually implemented using the %REPEAT...%FORFILES construct. It is an extremely powerful tool that can let you do absolutely anything you want (well, almost anything you want) to a fileset.

Accessing file information

As we just mentioned, every time the commands between the %REPEAT and the %FORFILES are executed, the MPEX variable "MPEXCURRENTFILE" is set to the name of the current file. (This is the fully-qualified filename, including the lockword [if any], group name, and account name.)

However, the filename is not the only piece of information available to you about the current file. In fact, all file attribute variables (see "[File attribute variables and functions](#)" in the MPEX Filesets chapter of this manual) for the current file are available by saying

```
RFILE.attrvar
```

For instance,

```
RFILE.CODE          contains the current file's filecode;
RFILE.FLIMIT        is set to the current file's file limit.
```

You can access these variables the same way that you access plain MPEX variables — in :IFs, in %CALCs, in %SETVARs, and using the "![*expression*]" variable substitution construct (see the [MPEX variables](#) chapter for more information).

For instance, our fileset `:PURGE` loop might very well look like:

```
%REPEAT
%>ECHO -----Purging !MPEXCURRENTFILE (![RFILE.SECTORS] sectors)
%>:PURGE !MPEXCURRENTFILE
%>FORFILES fileset
```

The string `![RFILE.SECTORS]` will be replaced by the value of the file attribute variable `RFILE.SECTORS`, which is the number of sectors the current file occupies. You might even say

```
%SETVAR SECTORSSAVED 0
%REPEAT
%>ECHO -----Purging !MPEXCURRENTFILE (![RFILE.SECTORS] sectors)
%>SETVAR SECTORSSAVED SECTORSSAVED+RFILE.SECTORS
%>:PURGE !MPEXCURRENTFILE
%>FORFILES fileset
%>ECHO Total: !SECTORSSAVED sectors saved
```

This uses `RFILE.SECTORS` in a `%SETVAR` statement to calculate the total number of sectors saved by the `%PURGE`.

One thing to bear in mind is that the values of these `RFILE` attribute variables are established only once. For example, if a file was in use when the fileset was evaluated, then `RFILE.OPENED` will be true for that file, even if it has since been closed.

"From" filesets and "To" filesets

What about a command like `:FCOPY` that handles more than one fileset. The "from-fileset" would be specified on the `%>FORFILES` command, and the name of each from-file would be available as `MPEXCURRENTFILE`. What about the "to-fileset"?

Well, say that we want to copy all files in the fileset `"S@.DATA"` into files in the fileset `"B@.BACKUP"` (i.e. files with the same name in the `BACKUP` group, but with the first `S` replaced by a `B`). We'd do it like this:

```
%REPEAT
%>:FCOPY FROM=!MPEXCURRENTFILE;TO=![OBJECTFILE("B@.BACKUP")];NEW
%>FORFILES S@.DATA
```

The new thing here is, of course, the

```
![OBJECTFILE("B@.BACKUP")]
```

construct. The `![...]` construct tells MPEX to execute the expression between brackets (`OBJECTFILE('B@.BACKUP')`) and substitute it into the command.

`OBJECTFILE` is a VESoft function, just like `MAX`, `MIN`, `STR`, `UPS`, etc. (please see [Appendix B](#) for more information on VESoft expressions, including functions). It takes as a parameter an "object fileset" and returns the name of a file in that fileset that corresponds to the current file in the current fileset.

In the above example, if

- the current file is `SMYPROG.DATA`, then
- `OBJECTFILE("B@.BACKUP")` will return the string `"BMYPROG.BACKUP"`.

Thus,

```
%REPEAT
%>:FCOPY FROM=!MPEXCURRENTFILE;TO=! [OBJECTFILE("B@.BACKUP")];NEW
%>FORFILES S@.DATA
```

will execute (for each file in `S@.DATA`) a command like

```
%;FCOPY FROM=SMYPROG.DATA;TO=BMYPROG.BACKUP;NEW
```

— just what a normal MPEX

```
%FCOPY S@.DATA,B@.BACKUP,NEW
```

command would do. In fact, MPEX's `%FCOPY` command is implemented using a `%REPEAT...%FORFILES` loop just like the one above.

Running programs and passing them input

One thing that you may want to do for each file is to run a program, passing to it input (perhaps including the current file's name and other attributes). For instance, to implement MPEX's `%EDIT` command, we'd like to `%RUN EDITOR.PUB.SYS` for each file and pass it a `/TEXT` of the current file, a command, and a `/KEEP`.

The way to do this is to say

```
%REPEAT
%>RUN EDITOR.PUB.SYS;INPUT="TEXT !MPEXCURRENTFILE";&
%> INPUT="the command to be executed";INPUT="KEEP";INPUT="EXIT"
%>FORFILES fileset
```

The `;INPUT=...` keyword causes `EDITOR.PUB.SYS` to be run with `STDIN` redirected to a temporary disk file to which MPEX has written the values of all the `;INPUT=...` strings.

For more information, see the section on the [;INPUT=...](#) keyword in the `%RUN` command documentation below.

Operating on spool filesets

The `%REPEAT...%FORFILES` construct can also be used for performing arbitrary commands on filesets of spool files. You must indicate that your fileset is a spool file fileset by appending a `":SPOOL"` to it on the `%FORFILES` command, i.e.

```
%REPEAT
...
%>FORFILES @.@.(SPOOL.OUTPRI=1):SPOOL
```

The commands between the %REPEAT and the %FORFILES will be executed once for each spool file in the spool file fileset. The MPEXCURRENTFILE variable will be set to "#0xxx" where "xxx" is the spool file number (e.g. "#01234"); the RFILE.*attrvar* fields will be available just as for normal files, and the spool file attributes will also be available by saying

```
SP00L.attrvar
```

(just like file attribute variables except that instead of being prefixed with "RFILE." they must be prefixed with "SP00L.").

For example, the following commands will copy all the \$STDLISTs in the DEV account into files in the SAVESP group whose filenames are "STD" followed by the spool file's spool file number:

```
%REPEAT
%>  RUN  SP00K5.PUB.SYS;&
%>    INPUT="TEXT !MPEXCURRENTFILE";&
%>    INPUT="COPY ALL,STD![SP00L.SP00LFILENUM].SAVESP";&
%>    INPUT="QUIT"
%>FORFILES $STDLIST.@.DEV:SP00L
```

Operating on temporary filesets

The %REPEAT...%FORFILES construct can also be used for performing arbitrary commands on filesets of temp files. You must indicate that your fileset is a temp fileset by appending a ":TEMP" to it on the %FORFILES command, i.e.

```
%REPEAT
...
%>FORFILES @.@.@:TEMP
```

When operating on 'temp' filesets, the MPEXCURRENTFILE variable contains the fully qualified name of the current TEMPorary file, but does not indicate in any way the file is a temp file. For example, the following is wrong:

```
%COMMENT --- Do NOT do this! This is an example of something that
%COMMENT --- doesn't work the way you might expect!
%REPEAT
%  PURGE !MPEXCURRENTFILE
%FORFILES @.FOOBAR.PROD:TEMP
```

This will purge all permanent files in the group FOOBAR.PROD that have the same name (fully qualified) filename as any of your temp files! What you should do (in this case) is:

```
%COMMENT --- Do this instead.
%REPEAT
%  PURGE !MPEXCURRENTFILE,TEMP
%FORFILES @.FOOBAR.PROD:TEMP
```

Only use commands that operate on temp files, and use whatever options you normally need to use to specify temp files.

%REPEAT...%FORJOBS

Syntax:

```
%REPEAT
%> command1
%> command2
...
%> command
%>FORJOBS userset
```

Examples:

```
%REPEAT
%>ECHO -----Processing job ![RJOB.FMTJOB] ![RJOB.FMTLOGON]
%>IF RJOB.INPRI=2 THEN
%>  :ALTJOB ![RJOB.FMTJOB];INPRI=3
%>ELSE
%>  :ABORTJOB ![RJOB.FMTJOB]
%>ENDIF
%>FORJOBS @.AP&WAIT
```

Similar to %REPEAT...%FORFILES, the %REPEAT...%FORJOBS construct allows you to execute an arbitrary sequence of commands on a VESoft userset (see the Usersets section of the SECURITY User Manual for details on usersets).

Note that you can use the `RJOB.attrvar` syntax to refer to the job attribute variables of each job much like you can use the `RFILE.attrvar` syntax in the %REPEAT...%FORFILES construct. For a list of attributes that can be used with this syntax, refer to the section "Job/Session Attributes" in this manual.

%REPEAT...%FORNUM

Syntax:

```
%REPEAT
%> command1
%> command2
...
%> command
%>FORNUM varname=startvalue,endvalue[,increment]
```

Examples:

```
%REPEAT
%>:BUILD TESTDAT!I; REC=-80,,F,ASCII
%>FORNUM I=1,9
```

The %REPEAT...%FORNUM construct is just like the FOR in C/PASCAL/SPL, PERFORM VARYING in COBOL, and DO in FORTRAN:

```
%REPEAT
%>:ABORTJOB #J!I
%>FORNUM I=955,970
```

will execute the %ABORTJOB sixteen times, setting the variable I to numbers from 955 to 970.

MPEX COMMAND REFERENCE: %REPEAT...%FORPROFILES

To increment I by a value other than 1 for each iteration, just say

```
%FORNUM I=startvalue,endvalue,increment
```

(e.g. %FORNUM I=0,100,10 will set I to 0, 10, 20, 30, ..., 100.)

%REPEAT...%FORPROFILES

Syntax:

```
%REPEAT
%> command1
%> command2
...
%> command
%>FORPROFILES profileset
```

MPEX's %REPEAT...%FORPROFILES construct allows you to perform an arbitrary set of actions on SECURITY profiles. Although the %SEC CHANGE command lets you change large groups of profiles, and %SEC LISTUSER lets you report on any set of profiles, these commands still have some limitations. %SEC CHANGE is limited to changing ALL profiles in the profileset to have the SAME value, and %SEC LISTUSER displays profiles using only one format.

Using %REPEAT...%FORPROFILES, however, you can change fields of each profile that qualifies with different values (which may even be based upon current values of other fields). Simple reports can be generated by using the %ECHO command to format and display the values of any field within the profile.

Within the loop itself, profile field values are referenced using the same syntax as in %REPEAT...%FORFILES, namely, use ! [PROFILE.*attribute*] as part of the command (refer to the USER PROFILE ATTRIBUTES chapter of the SECURITY manual for a description of these attributes). The variable MPEXCURRENTPROFILE will contain the key value for the current profile, in *session,user.account* format, for use in a %SEC CHANGE command.

For example, let's say you have a user defined field that contains a three-digit department code and you want to change them to be four-digits by adding a "1" to the beginning of each code.

The following %REPEAT...%FORPROFILES loop will do just that:

```
%REPEAT
%SEC CHANGE !MPEXCURRENTPROFILE;DEPT=!"1"+PROFILE.UF('DEPT')]
%FORPROFILES @,@.@(UF('DEPT')<>'')
```

%REPEAT...%FORRECS

Syntax:

```
%REPEAT
%> command1
%> command2
...
%> command
%>FORRECS varname=fopenparms
```

Examples:

```
%REPEAT
%>RUN MYPROG1; INPUT=PROCDATE
%>FILE DATAFILE=D!PROCDATE.DATA
%>RUN MYPROG2
%>FORRECS PROCDATE=MYFILE,OLD
```

The %REPEAT...%FORRECS construct performs a sequence of MPEX commands once for each record of a file, with a given variable (for example, "CURRENTREC") set to each file record:

```
%REPEAT
%>ECHO Processing !CURRENTREC
%>RUN MYPROG;INFO="!CURRENTREC"
%>FORRECS CURRENTREC=DATAFILE,OLD
```

will do the commands between "%REPEAT" and "%FORRECS" once for each record in the file DATAFILE, setting the variable CURRENTREC to the contents of the current record.

%RESUMEJOB

Syntax:

```
%RESUMEJOB [userset]
             [;NOVERIFY]
             [;QUIET]
```

Examples:

```
%RESUMEJOB @.AP
%RESUMEJOB
%RESUMEJOB REPORTJ,@.@
%RESUMEJOB #J1234+#J1235
```

The %RESUMEJOB command is an enhanced version of MPE's :RESUMEJOB command. It allows you to specify which jobs to resume using VESoft usersets (see the Usersets section of the SECURITY User Manual). By default, the userset is "SUSP" or all suspended jobs (in fact, "&SUSP" is appended to the userset supplied since it doesn't make sense to try and :RESUMEJOB a job that is not suspended).

%RETURN

Syntax:

```
%RETURN
```

Examples:

```
<< sample MPEX command file: >>
PARM FILE
IF NOT FEXISTS("!FILE") THEN
    ECHO Error: !FILE does not exist.
    RETURN
ENDIF
...
```

MPEX COMMAND REFERENCE: %RUN

The %RETURN command exits the currently-executing UDC or command file; in the example above, it's used to exit a command file in case the FILE parameter turns out to be invalid.

This sort of thing — exiting command files and UDCs in case of an error condition — is what %RETURN is most often used for.

%RUN

Syntax:

```
%RUN {progfile|pin} [, entry]
      [;mperunparms]
      [;GOON]
      [;INMSG]
      [;INPUT= inline] [;INPUT=...]
      [;KILL]
      [;NEW]
      [;NOACTIVATE]
      [;NOACTSONTERM]
      [;PRI= {CS|DS|ES}]
      [;STDIN= {fileref|$NULL| "file-eq-parms"}]
      [;STDLIST= {fileref|$NULL| "file-eq-parms"}]
```

Examples:

```
%RUN MYPROG; MAXDATA=30000; LIB=G
%RUN BIGHOG; PRI=ES; GOON
%RUN MYPROG; STDIN="TEMPMSG;REC=, ,V,ASCII;DEL";&
      STDLIST="OUTFILE,OLDTEMP;SHR;GMULTI;TEMP"
%RUN SEGDVR.PUB.SYS; INPUT="USL U"; INPUT="AUXUSL AU";&
      INPUT="COPY SEGMENT,MYSG"; INPUT="EXIT"
```

MPEX's %RUN command fully emulates all the parameters of the MPE :RUN command (;LIB=, ;PARM=, ;INFO=, ;MAXDATA=, etc.), and implements a few of its own (see "[MPEX process handling](#)" in the MPEX Special Topics chapter for more information — these features are fairly advanced, and are probably not for beginning users):

- ;PRI={CS|DS|ES}, which lets you indicate the queue in which the son process is to run (just like the ;PRI=... parameter on a :JOB card works for a job).
- ;GOON ("go on"), which means that MPEX shouldn't suspend when it creates the son process, but rather go on processing with both MPEX and the son being active.
Don't confuse this keyword with the %GOON command prefix. Programs run using :RUN ...;GOON are NOT reported by the %SHOWGOON command.
- ;NOACTSONTERM, which means that MPEX should not be automatically re-activated when the son process terminates. This can sometimes be useful in conjunction with the ;GOON keyword; however, if you use ;NOACTSONTERM without using ;GOON, you're liable to get hung up when the son process terminates and does not re-activate MPEX.
- ;STDIN=.../STDLIST=... "parms", which let you specify not just the filename for the son's STDIN/STDLIST, but a full file equation (except for the word ":FILE"), including options such as ;DEL, ;SHR, ;REC=, etc.

See "[Command I/O redirection](#)" for another way to re-direct STDIN and STDLIST.

- `;INPUT=` (and `;INMSG`), which let you pass data as input to the son process without building a special `STDIN` file. MPEX will build the `STDIN` file for you, writing the values of all the `;INPUT=...` parameters to it; the son process will then be run with that `STDIN` file. (See also "[%RUNCREATE](#), [%RUNINPUT](#), [%RUNACTIVATE](#)", below).
- `;NOACTIVATE`, which creates the son process without activating it.
- `;NEW`, which commands MPEX to create a new process, even if another son process with the same name exists and is ready to be re-activated.
- `;KILL`, which tells MPEX to kill the son process when it's done (in case it suspends rather than terminates).
- Instead of specifying the name of the program to be run, you may specify the PIN of an already existing son process; in that case, that process will be re-activated.
- If you have SM capability or are AM of the account in which the program you are %RUNing resides, and the program file has a lockword, MPEX automatically inserts lockwords for you.

%RUNCREATE, %RUNINPUT, %RUNACTIVATE

Syntax:

```
%RUNCREATE program [,entry] [;mpexrunparms]

%RUNINPUT inputline

%RUNACTIVATE
```

Examples:

```
%RUNCREATE MYPROG,PROCFILES; PARM=1; PRI=CS
%REPEAT
%>RUNINPUT !MPEXCURRENTFILE
%>FORFILES @.DATA
%RUNINPUT EXIT
%RUNACTIVATE
```

MPEX's `%RUN...;INPUT=...` lets you run a program and pass input lines to it (without having to specially build a `STDIN` file). What if, however, you want to pass many input lines (more than can fit on one `%RUN` command)? This is particularly relevant for `%REPEAT...%FORFILES`, with which you might want to pass a program one or more lines of input for each file in the fileset.

MPEX's `%RUNCREATE`, `%RUNINPUT` and `%RUNACTIVATE` commands let you do this — `%RUNCREATE` creates a son process but doesn't actually run it; `%RUNINPUT` writes a line of input to its `STDIN` file; `%RUNACTIVATE` actually activates it and waits for it to finish executing all the commands you passed to it using `%RUNINPUT`.

For instance,

```
%RUNCREATE KSAMUTIL.PUB.SYS
%REPEAT
%>RUNINPUT KEYINFO !MPEXCURRENTFILE
%>FORFILES @.@.AP(ISKSAM and NOT ISKSAMXL)
%RUNINPUT EXIT
%RUNACTIVATE
```

MPEX COMMAND REFERENCE: %SAVEJOB, %SHOWSAVED, %DOSAVED

This will run `KSAMUTIL.PUB.SYS`, pass it a `KEYINFO` command for each file in the fileset (which automatically recovers KSAM files that were open during a system failure), and then an `EXIT` command, and finally activate `KSAMUTIL.PUB.SYS` to let it execute all those commands.

Note that the program is not activated until the `%RUNACTIVATE` — the `%RUNINPUT` commands don't actually "execute" the commands but rather buffer them up in the son process' `STDIN` file to be executed when the `%RUNACTIVATE` is done.

`%RUNCREATE`, `%RUNINPUT` and `%RUNACTIVATE` are actually shorthand for "`%RUN progname ;NOACTIVATE;NEW;INMSG`", "`%RUN progname;INPUT=...`" and "`%RUN progname;KILL`" respectively — all of which are documented MPEX `:RUN` command enhancements.

One important point about these commands is the fact that they use a message file to pass your input to the program being run. If you forget to `%RUNINPUT` an `EXIT` command of some sort, then the program will merely suspend after it is done executing the commands you gave it instead of terminating (This is similar to how the `%SPOONFEED` command works). The problem that this creates is that control is never returned to MPEX, so it appears that our program hangs. You should press `[BREAK]` and type `:ABORT` to terminate MPEX and the "suspended" program.

%SAVEJOB, %SHOWSAVED, %DOSAVED

Syntax:

```
%SAVEJOB userset; FILE=savefile
        [;NOVERIFY]
        [;QUIET]

%SHOWSAVED; FILE= savefile

%DOSAVED;FILE=savefile
        [;NOVERIFY]
        [;QUIET]
        [;MUSTEXEC]
        [;mpeschedparms]
```

Examples:

```
%SAVEJOB WAIT+SCHED; FILE=SAVEDJBS
%SAVEJOB @.PROD; FILE=PRODJOB; NOVERIFY
%SAVEJOB @.-EXEC; FILE=ALLJOBS; QUIET
%SAVEJOB #J1234+#J1235+#J1236; FILE=CRITJOBS

%SHOWSAVED;FILE=PRODJOB

%DOSAVED;FILE=SAVEDJBS
%DOSAVED;FILE=PRODJOB; NOVERIFY; AT=23:30
%DOSAVED;FILE=ALLJOBS; QUIET; MUSTEXEC
%DOSAVED;FILE=CRITJOBS; IN=0,0,5
```

For security reasons, these commands can only be executed by SM capability users.

You need to shut your system down, but there are a bunch of jobs in the WAIT and SCHED state. What do you do?

You *could* try writing down the names of all of the jobs and then re-:STREAM them after you bring the system back up, but the :SHOWJOB command only shows you how the job logs on, not the name of the file that was :STREAMed (sometimes they match, but...). What if the job file doesn't even exist anymore? If you use STREAMX, how are you going to know what answers were entered to any :PROMPTs the jobs contain?

The %SAVEJOB command saves the \$STDIN files for all (or selected) jobs, so that you can later re-submit them via the %DOSAVED command (described below)! Since the \$STDIN file — not the job file — is what MPE actually executes, the saved job (when re-submitted later) will be exactly the same (except for the job number) as it was before you saved it:

- all of the job-card parameters (like ;INPRI=, etc.) will be preserved;
- if you use SECURITY, any necessary \$BATCH-VEPROFILE job authorizations will still be in effect;
- all STREAMX parameter substitutions were done before the \$STDIN was created, so they won't be lost;
- if the job was in a SCHED state at the time it was saved, the scheduling information will also be preserved.

Here's how it works:

```
%SAVEJOB WAIT+SCHED; FILE=SAVEDJBS
-----Will SAVEJOB #J41, DISCLEAN,MANAGER.SYS,JOB
-----Will SAVEJOB #J38, SEND,MANAGER.LIST,VESOFT
-----Will SAVEJOB #J33, VEAUDIT,MANAGER.VESOFT,PUB
-----Will SAVEJOB #J34, NIGHTLY,MGR.PROD,PUB
OK to process these jobs (y/n)? Y
-----SAVEJOB #J41, DISCLEAN,MANAGER.SYS,JOB
-----SAVEJOB #J38, SEND,MANAGER.LIST,VESOFT
-----SAVEJOB #J33, VEAUDIT,MANAGER.VESOFT,PUB
-----SAVEJOB #J34, NIGHTLY,MGR.PROD,PUB
```

Like the %ABORTJOB, %TELL, %ALTSCHED and other MPEX jobset-handling commands, %SAVEJOB allows you to specify which job(s) to save using VESOFT usersets (documented fully in the usersets section of the SECURITY User Manual).

If you want %SAVEJOB to save the jobs without asking you if it's OK, use the ;NOVERIFY keyword.

If you don't even want %SAVEJOB to show you the jobs as it saves them, use the ;QUIET keyword.

At this point, you can shut down your system or perhaps abort these jobs via the MPEX %ABORTJOB command (%SAVEJOB does not abort the existing job) and wait for the currently executing jobs to finish before shutting down.

Viewing saved jobs (%SHOWSAVED)

%SAVEJOB copies the jobs to the "savefile" in a special format (and creates the "savefile" as a PRIV file — you must use MPEX to %PURGE it). To see what jobs are in a "savefile", use the %SHOWSAVED command:

```
%SHOWSAVED;FILE=SAVEDJBS
:JOB DISCLEAN,MANAGER.SYS,JOB;PRI=DS;OUTCLASS=LP,3,1
    DATE=01/29/92;AT=03:00
:JOB SEND,MANAGER.LIST,VESOFT;HIPRI;PRI=CS;OUTCLASS=LP,2,1
    DATE=01/29/92;AT=02:30
:JOB VEAUDIT,MANAGER.VESOFT,PUB;PRI=DS;OUTCLASS=LP,1,1
    DATE=01/29/92;AT=22:00
:JOB NIGHTLY,MGR.PROD,PUB;HIPRI;PRI=DS;OUTCLASS=LP,1,1
    DATE=01/29/92;AT=00:01
```

Re-submitting saved jobs (%DOSAVED)

After you bring the system back up (or whenever you like) you can re-submit saved jobs by typing:

```
%DOSAVED;FILE=SAVEDJBS
```

By default, this will display the job logon and scheduling parms (if any) for each job in the SAVEDJBS file, and ask if you want to submit that job:

```
Submit DISCLEAN,MANAGER.SYS,JOB [date=01/29/92;at=03:00] (y/n)? Y
#J54
Submit SEND,MANAGER.LIST,VESOFT [date=01/29/92;at=02:30] (y/n)? Y
#J55
Submit VEAUDIT,MANAGER.VESOFT,PUB [date=01/29/92;at=22:00] (y/n)? Y
#J56
Submit NIGHTLY,MGR.PROD,PUB [date=01/29/92;at=00:01] (y/n)? N
```

Like %SAVEJOB, you can use the ;NOVERIFY to submit job(s) without prompting or ;QUIET to submit them without even listing them.

When you re-submit saved jobs, %DOSAVED simply re-submits the jobs with the originally scheduled date and time. Generally, this is OK; however, you may end up in the situation where a saved job was scheduled to run specifically at a time when the jobs were not on the system (i.e., if you had to take the system down, and a job was scheduled to run during the time the system was down). In this case, an error message is displayed regarding an attempt to stream a job prior to the current date and time.

There are two methods you can use to avoid this problem

- Specify the keyword ;MUSTEXEC, which signals to our software that if the time for a job to launch has passed, stream the job immediately.
- Supply new scheduling parameters with the %DOSAVED command (like ;AT=, ;DAY=, ;DATE= or ;IN=) to override those that were in effect when the job was saved;

e.g.

```
%DOSAVED;FILE=SAVEDJBS; DATE=01/31/92
Submit DISCLEAN,MANAGER.SYS,JOB [date=01/31/92] (Y/n)? N
Submit SEND,MANAGER.LIST,VESOFT [date=01/31/92] (Y/n)? N
Submit VEAUDIT,MANAGER.VESOFT,PUB [date=01/31/92] (Y/n)? N
Submit NIGHTLY,MGR.PROD,PUB [date=01/31/92] (Y/n)? Y
#J58
```

Any scheduling parameters specified with %DOSAVED will completely replace all scheduling parameters that were "saved" with the job; if the job was not scheduled when it was saved (i.e. it wasn't in a SCHED state), the %DOSAVED scheduling parms will not be applied to it.

If you want to PURGE the savefile (%DOSAVED does not automatically PURGE it), simply use the MPEX %PURGE command — MPE's :PURGE command doesn't purge PRIV files.

Important note: There is no need to save schedule if you use STREAMX's scheduling — see the SECURITY User Manual for details.

%SCHEDULE

Syntax:

```
%SCHEDULE "scheduleparms", "[joblogon] [;jobparms]", command
```

Examples:

```
%SCHEDULE "AT=3:00;DAY=TUE",, COBOL AP@.SOURCE, AP@.PUB
%SCHEDULE "AT=21:00", ";OUTCLASS=,3", RUN MYPROG;PARAM=1234
%SCHEDULE "AT=1:00", "MANAGER.SYS", PURGE LOG####.PUB.SYS
```

The %SCHEDULE command makes MPEX submit a scheduled job that will execute the given MPEX command.

- You can use any MPE scheduling parameters (AT=, DAY=, etc.).
- You may (optionally) indicate that the job is to be submitted under somebody else's user ID or with some special job parameters.
- You may execute either an MPEX command or a normal MPE command (RUN, STORE, a UDC, etc.); if the MPE command has the same name as an MPEX command (like FCOPY), precede it with a ":".
- All the file equations that are in effect in your session and the current value of your HPPATH variable will be copied to the created job.
- You may indicate who MPEX should notify of the job's completion by setting the MPEXTELLBACK variable.
- You may indicate how MPEX should inform you of the job's completion by setting the MPEXTELLBACKCMD variable.

(See the %SUBMIT and "Batch execution ('\$ prefix)" in the MPEX Commands chapter for more information.)

To execute an MPEX command in a job stream, you can simply say

```
%$COBOL AP@.SOURCE, AP@.PUB
```

The "\$" tells MPEX to perform the command offline — MPEX will automatically generate a job stream that executes the above command.

It'll also copy all the file equations (such as :FILE COPYLIB=...) that you may have in your session into the job, so that the job environment is as close as possible to your session environment.

The %SCHEDULE command simply extends this capability to allow you to specify scheduling and :JOB card parameters for the job stream. Saying

```
%$COBOL AP@.SOURCE, AP@.PUB
```

will execute "COBOL AP@.SOURCE, AP@.PUB"

- immediately,
- under your own logon id,
- with the job parameters ";OUTCLASS=,1".

What if you want to submit it at 3 AM Tuesday? Just say:

```
%SCHEDULE "AT=3:00;DAY=TUE",, COBOL AP@.SOURCE, AP@.PUB
```

The first parameter ("AT=3:00;DAY=TUE") specifies the :STREAM command scheduling parameters; as the second parameter is omitted, logon ID and job parameters will be defaults.

If you say:

```
%SCHEDULE "AT=3:00;DAY=TUE", "MGR.PROD;OUTCLASS=SLOWLP,3", &
COBOL AP@.SOURCE, AP@.PUB
```

then it'll not only define the scheduling parameters but also the logon (MGR.PROD) and job card parameters (;OUTCLASS=SLOWLP,3).

Saying

```
%SCHEDULE "AT=3:00;DAY=TUE", ";OUTCLASS=SLOWLP,3;INPRI=4", &
COBOL AP@.SOURCE, AP@.PUB
```

will only change the job card parameters (;OUTCLASS=SLOWLP,3;INPRI=4); the logon will be the default (your own logon ID).

Saying

```
%SCHEDULE "AT=3:00;DAY=TUE", "MGR.PROD", COBOL AP@.SOURCE, AP@.PUB
```

will only change the logon ID (MGR.PROD) and leave the job card parameters at their default setting (;OUTCLASS=,1).

Of course, if you try to submit something under somebody else's logon ID, MPEX will prompt you for the passwords (unless you have SM capability or AM capability in your own account).

%SEC

Syntax: `%SEC command`

Examples: `%SEC LISTUSER @.@`
`%SEC ADD JAY,MGR.SALES; ASKPASS`

The %SEC prefix allows SECURITY users to enter SECURITY maintenance commands within MPEX. Please see "Command-Driven SECURITY Maintenance" in the SECURITY User Manual for details.

%SECURE

Syntax: `%SECURE fileset`
`[;KEEPAMDATES]`

Examples: `%SECURE @.@.PROD-TEST@.@.PROD(ISRELEASED)`
`%SECURE @.@.@ &`
`(ISPROG and ISRELEASED and DIRGROUP.CAP("PM"))`

The MPEX %SECURE command lets you

- %SECURE entire filesets
- even if you're not the file's creator (all it takes is Read, Write, and eXecute access — since with Read, Write, and eXecute, you can do anything to the file [including purging it], this is quite safe); additionally,
- the ;KEEPAMDATES keyword allows you to secure a file (or fileset) without changing that file's last Access and Modify dates.

As you know, **:RELEASEing a file is very dangerous to system security**. When you :RELEASE a file, you are letting anybody do anything to it — read it, write to it, even purge it! There's no way of :RELEASEing a file for read access only or to a particular user or set of users.

If you :RELEASE a (non-privileged) file, you lose absolutely all security with respect to it.

This is why the %SECURE command can be so useful. Chances are that there are hundreds of :RELEASEd files on your system, :RELEASEd from ignorance (most people don't know how wide-open the :RELEASE command leaves them) or laziness.

The only way you can %SECURE them all in one fell swoop is with MPEX. You could always find them with the %LISTF command:

```
%LISTF @.@.PROD(ISRELEASED)
```

but you could always just **:SECURE** them all:

```
%SECURE @.@.PROD(ISRELEASED)
```

(the **ISRELEASED** selection condition will find only those files that are actually **:RELEASED**, and thus avoid re-**:SECURE**ing files that are already **:SECURED**).

If for some reason, you need some files to remain **:RELEASED**, you can easily tailor your selection conditions to exclude them from the **%SECURE**:

```
%SECURE @.@.PROD-TEST@.@.PROD(ISRELEASED and NOT ISPRIV)
```

will **:SECURE** all the **:RELEASED** files in the **PROD** account EXCEPT for databases (**NOT ISPRIV**) and those files that start with **TEST**. (As we discuss in the **%RELEASE** command, databases are the only files that it's often a good idea to **:RELEASE**.)

Vital for system security!

There's one very important way in which **:RELEASED** files can **ENTIRELY COMPROMISE YOUR SYSTEM SECURITY**.

If you have any

- program files
- that are **:RELEASED**
- and reside in groups that have PM capability,

then

ANY USER — WITHOUT ANY SPECIAL CAPABILITIES — WILL BE ABLE TO ACQUIRE SM CAPABILITY!

For obvious reasons, we won't tell you how he could do this; however, it's vital that all the files that match the conditions shown above must be **:SECURED** (with NO exceptions). With MPEX, nothing could be easier:

```
%SECURE @.@.(ISPROG and ISRELEASED and DIRGROUP.CAP("PM"))
```

Just another example of the power of MPEX selection conditions.

%SET CAPABILITY

Syntax:

```
%SET CAPABILITY, command, {SM|AM|AL}
```

Examples:

```
%SET CAPABILITY, PURGE, AM
%SET CAPABILITY, RELEASE, SM
```

MPEX does not in any way endanger your system security — anything "dangerous" you could do in MPEX can be done with simple MPE commands (although it would take longer!).

However, when MPEX makes it easier for you to do some things, it has no way of knowing that what you ask it to do is what you really want it to do.

If you say

```
%RELEASE @
```

MPEX will faithfully **:RELEASE** all the files in your group (assuming you have Read/Write/Execute access to the files). Never mind the fact that this will remove ALL security from those files — you asked it to **:RELEASE** them, and that's exactly what it'll do.

Because of this, a system manager might want to restrict access to some MPEX commands — usually **%PURGE** and **%RELEASE** — to people with certain capabilities. For instance, he might put into the **MPEXMGR.PUB.VESOFT** file (which always gets executed whenever anybody enters MPEX — see "[Specifying commands to be executed every time MPEX is entered](#)" in the Running MPEX section) the commands:

```
%SET CAPABILITY,PURGE,AM
%SET CAPABILITY,RELEASE,SM
```

This means that

- Only users with SM capability will be allowed to use the **%RELEASE** command.
- Only users with AM capability will be allowed to use the **%PURGE** command. Users with AM only will be able to **%PURGE** files only in their own accounts — users with SM will be able to **%PURGE** files anywhere.

The capabilities that you can specify are **SM**, **AM** or **AL** (Account Librarian).

We advise, however, that you use **%SET CAPABILITY** sparingly; naturally, we think that most MPEX commands are just too useful to deny access to. Please see the chapter on "[Restricting MPEX commands to only work on certain files](#)" in the Running MPEX section in this manual. There, we explain how you can set up your own commands with your own restrictions — for instance, you could easily build a **%PURGE** that will not purge databases or KSAM files or an **%ALTFILE** that will only work for files in your logon group.

%SET CREATORPROTECT

Syntax:

```
%SET CREATORPROTECT
```

MPEX's **%ALTFILE...;CREATOR=** and **%COPY...;CREATOR=** keywords let you change a file's creator ID. This can be a very useful feature (see the documentation on [%ALTFILE...;CREATOR=](#) for more information) in many instances, including changing the creator IDs of files whose creators have been purged. Normally (on non-POSIX systems), MPEX allows anybody who has Read, Write, and eXecute access to a file to **%ALTFILE** its creator ID, since with that sort of access the user can do anything to the file anyway.

Some users need creator IDs for their own internal auditing purposes and therefore don't want to let people change files' creator IDs. The **%SET CREATORPROTECT** command lets system managers impose this restriction if they so wish.

MPEX COMMAND REFERENCE: %SET DATE

If you put a "SET CREATORPROTECT" command into your MPEXMGR.PUB.VESOFT (see ["Specifying commands to be executed every time MPEX is entered"](#) in the Running MPEX section), then only users with SM capability and users with AM capability working on files in their own accounts will be allowed to do an %ALTFILE...;CREATOR=... or a %COPY...;CREATOR=... (of course, non SM/AM users can still use any of the other %ALTFILE and %COPY keywords). Also, if any non-SM non-AM user uses the %COPY...;KEEPATTR command, the creator ID will not be kept by the copy, but will instead be set to the copying user's name.

%SET CREATORPROTECT will still permit SM users and AM users working on files in their own accounts to use %ALTFILE...;CREATOR=, %COPY...;CREATOR=... and %COPY...;KEEPATTR without restriction.

POSIX Due to changes in MPE regarding file security, MPEX enforces %SET CREATORPROTECT on note: POSIX based systems. This is to maintain compatibility with MPE's :ALTFILE command which is designed to let SM and AM users change the OWNER of a file. See the [%ALTFILE command](#) for more details.

%SET DATE

Syntax: `%SET DATE,{MDY|DMY|YMD}`

Examples: `%SET DATE,MDY`

By default, MPEX dates — in selection conditions, in the %CALC, %IF, %SETVAR, etc. commands, and so on — are expected to be in MM/DD/[YY]YY (2- or 4-digit year, with or without slashes) format.

You can change this by saying

```
%SET DATE,DMY
```

to indicate that input dates will be in DD/MM/[YY]YY format, or

```
%SET DATE,YMD
```

to indicate that input dates will be in [YY]YY/MM/DD format.

Note that while most commands do not make use of this setting to determine how to display date values, a few commands do use this setting. The commands that do make use of this setting have a note to that effect in their command description. If you want to output a date in a specific format (in a %CALC command or your own %LISTF template file), you can use the CDATEMDY, CDATEDMY, and CDATEYMD functions or the built-in formatting capability of the WRITELN and STRWRITE functions. (see [Appendix B](#) for more details).

%SET DEFAULT

Syntax:

```
%SET DEFAULT, command, {? | ! | $}
```

Examples:

```
%SET DEFAULT, PURGE, ?
%SET DEFAULT, COBOL, $
```

By default, all MPEX commands are executed online, immediately as you type them. You can always prefix any MPEX command with a "\$", e.g.

```
__$COBOL AP@.SOURCE, AP@.PUB
```

to perform it offline, or prefix it with a "?", e.g.

```
 $?PURGE @.TEST
```

to verify the %PURGE for every file before purging that file (MPEX will ask you "OK to process X.TEST (y/N)? ", and if you answer N, the operation won't be done on this file).

You might want to make either "\$" execution or "?" execution the default for some commands; for instance,

```
%SET DEFAULT, PURGE, ?
%SET DEFAULT, COBOL, $
```

to indicate that all %PURGEs are to be done with verification (for safety's sake) and all %COBOLs are to be done offline (to avoid tying up your terminal).

When these defaults are set, you can still say

```
%!PURGE K#####.@
%!COBOL AP010.SOURCE, AP010.PUB
```

to execute the commands ONLINE with no verification (that's what the "!" prefix does); however, the DEFAULT will be "?" for %PURGE and "\$" for %COBOL.

Remember, in all cases you can execute each command any way you like (online, offline or with verification) with one extra keystroke; %SET DEFAULT merely indicates which mode is to be used if you specify neither "!", "?", nor "\$".

If you use the %SET DEFAULT command, you'll probably want to embed it into

- the MPEXMGR.PUB.VESOFT file, so it'll be automatically executed every time anybody runs MPEX (and will thus be in effect on a SYSTEM-wide basis),
- the MPEXMGR.PUB file of your own account, so it'll be automatically executed every time anybody in that account runs MPEX, and/or
- the MPEXMGR file of your own group, so it'll be automatically executed every time anybody in that group runs MPEX.

See "[Specifying commands to be executed every time MPEX is entered](#)" in the Running MPEX section for more information.

%SET GOONMAXPRI

Syntax: `%SET GOONMAXPRI,{CS|DS|ES}`

Examples: `%SET GOONMAXPRI,CS`
`%SET GOONMAXPRI,ES`

The %GOON command (documented earlier in this manual) gives MPEX users an easy, convenient way to execute commands, run programs, etc. as son processes. By default, %GOON processes execute in the DS queue (which gives them about the same priority as batch jobs, on most systems) but users may change this by setting the VESOFTGOONPRI variable.

Although this doesn't allow users to do anything that MPE security would normally prevent them from doing, it does make it much easier for one user to create many active son processes in the CS queue. The system manager might wish to restrict %GOON processes to the DS (or even the ES) queue.

To do this, simply add a line to your MPEXMGR.PUB.VESOFT file that says something like:

```
%SET GOONMAXPRI,DS
```

Now, even if the user sets the VESOFTGOONPRI variable to "CS", his %GOON processes will run in the DS queue.

Once a %SET GOONMAXPRI command has been executed within your current run of MPEX, you can not change it to a higher queue until you exit MPEX and re-run it; i.e. if you set the GOONMAXPRI to DS (as in the example above), you will not be able to change it back to CS unless you first exit MPEX (but you could change it to ES).

%SET [NO]CMDTRACECHECK

Syntax: `%SET [NO]CMDTRACECHECK`

Examples: `%SET CMDTRACECHECK`
`%SETJCW MPEXCMDTRACE = 1`

MPEX, STREAMX and VEMENU all let you trace execution of commands using the MPEXCMDTRACE JCW; if you set it to 1, each command is output before it is executed, which can help you debug complicated command files, job streams or menus.

It takes time to check this variable and this can add up if you have a very complicated command file or menu (one of our users sent us a menu file that had about 100 lines per caption!), especially if you are on a heavily loaded system.

Therefore, we have two commands, usable from MPEX, STREAMX, or VEMENU – SET NOCMDTRACECHECK, which turns off MPEXCMDTRACE checking, and SET CMDTRACECHECK, which turns it on. Checking is on by default for MPEX and off by default for STREAMX and VEMENU, but these SET commands can be used to alter this setting.

%SET [NO]VARTRACE

Syntax: `%SET [NO]VARTRACE`

Examples:

```
%SET VARTRACE
%XEQ MYCMDFIL
%SET NOVARTRACE
```

You can write MPEX command files that do a lot of different things (our users constantly surprise us with the creative use to which they put our software). The `%SET VARTRACE` command was implemented to help you debug these command files.

```
%SET VARTRACE
```

tells MPEX to echo to `$STDLIST` the name of any variable whose value changes, the old and new values of that variable, and the name of the command file or UDC which changed it (`$STDIN`, if the variable was changed by an interactive command).

Using `%SET VARTRACE` is much easier than adding a bunch of `"ECHO"` commands to the command file, and it shows all variables that are changed (including "local" variables), not just the ones you thought to `"ECHO"`.

```
%SET NOVARTRACE
```

turns the tracing off.

For example, suppose you have a command file named `TEST.WORK.VESOFT` that looks like this:

```
SETVAR TEST_VAR "tadpole"
SETVAR TEST_VAR "frog"
SETLVAR TEST_LOCAL_VAR "caterpillar"
SETLVAR TEST_LOCAL_VAR "butterfly"
```

You could watch these variables change using `%SET VARTRACE`, like this:

```
%SET VARTRACE
%TEST
[TEST.WORK.VESOFT] TEST_VAR := tadpole
[TEST.WORK.VESOFT] TEST_VAR (tadpole) := frog
[TEST.WORK.VESOFT] TEST_LOCAL_VAR [local] := caterpillar
[TEST.WORK.VESOFT] TEST_LOCAL_VAR [local] (caterpillar) := butterfly
%SET NOVARTRACE
```

%SETLVAR

Syntax: `%SETLVAR varname expression`

Examples:

```
%SETLVAR COUNTER 1
%SETLVAR ANSWER UPS(ANSWER[0:1])
%SETLVAR JULIANDAY TODAY-DATEBUILD(HPYEAR,1,1)+1
```

The %SETLVAR command evaluates an expression and assigns its value to an MPEX local variable.

Please see "[Local variables](#)" in the MPEX Variables chapter of this manual for more details.

%SETVAR

Syntax: `%SETVAR varname expression`

Examples:

```
%SETVAR NUMFILES NUMFILES+1
%SETVAR FRECSIZE -VEFINFO(FNAME+".DEV.AP").RECSIZE
%SETVAR NUMBERISVALID BETWEEN(NUMINPUT,-5,15+NUMWIDGETS)
%SETVAR FOURTHREC VEFREADDIR("DATAFILE,OLD",3)
%SETVAR C SQRT((A)+(B))
%SETVAR LASTDAYOFLASTMONTH DATEBUILD(HPYEAR,HPMONTH,1)-1
%SETVAR LASTFRIDAY TODAY-DATEDAYOFWEEK(TODAY-6)
%SETVAR NEXTFRIDAY TODAY+7-((HPDAY+1) MOD 7)
%SETVAR JULIANDAY TODAY-DATEBUILD(HPYEAR,1,1)+1
```

The %SETVAR command evaluates an expression and assigns its value to an MPEX variable (see the [MPEX variables](#) chapter). This variable can later be substituted into MPEX commands, and used in %IFs, %WHILEs, %CALCs and other %SETVARs.

If the variable "*varname*" exists, its old value will be overwritten with the new; if it doesn't exist, it will be created.

The expression can be any VESOFT expression, not just an ordinary MPE expression. The syntax of VESOFT expressions is described fully in the Appendix.

VESOFT expressions support real numbers, dates and times as well as integers, booleans and strings. Unfortunately, MPEX variables can only be integers, booleans, and strings (because of the restrictions imposed by MPE/iX with which we must be compatible). If the expression returns a real number, a date or a time, that value will be nicely formatted into a string, and the variable "*varname*" will be a string variable.

%SHOW

Syntax: `%SHOW`

Examples: `%SHOW`
`SET CAPABILITY,RELEASE,SM`
`SET DEFAULT,PURGE,?`
`SET DATE,MDY`

Shows the values of all the

- `%SET CAPABILITY` (capability restrictions on MPEX commands),
- `%SET DEFAULT` (default modes of execution for MPEX commands), and
- `%SET DATE` (input date format, MDY vs. DMY vs. YMD)

options that have been set.

%SHOWALARM

Syntax: `%SHOWALARM [;ALL]`
`[;FROM=mpeuserset]`

Examples: `%SHOWALARM`
`%SHOWALARM;ALL`
`%SHOWALARM;FROM=@.DEV`

`%SHOWALARM` shows information about the alarms you set via the `%ALARM` command. For more information, see the documentation on [%ALARM](#).

%SHOWGOON

Syntax: `%SHOWGOON`

Examples: `%SHOWGOON`

FILENAME	[STATUS]	COMMAND
VE053	[done]	PRINT @.SOURCE;SEARCH="SOC-SEC-RATE";NUM
VE073	[active]	ALTFILE @.@.DEV; XLTRIM; KEEPAMDATES
VE094	[active]	COBOL MYPROG.SOURCE,=.PUB

`%SHOWGOON` displays the status of all of your `%GOON` processes. For more information, please see the [%GOON command](#) in this manual.

%SHOWJOB

Syntax:

```
%SHOWJOB [mpe-showjob-parameters]
           [;JOB=@A]
           [;NOSEC]
```

Examples:

```
%SHOWJOB JOB=@A
%SHOWJOB SCHED;NOSEC
%SHOWJOB JOB=@.SYS;*LP
```

```
%SHOWJOB
```

JOBNUM	STATE	IPRI	JIN	JLIST	INTRODUCED	JOB NAME
#S2	EXEC		20	20	WED 7:41A	CHUCK,MANAGER.SYS
#J3	EXEC		10R	LP	WED 7:43A	BACKG,MANAGER.VESOFT
#S4	EXEC	QUIET	3	3	WED 8:05A	VLADIMIR,MANAGER.VESOFTD
#S6	EXEC		5	5	WED 8:10A	RON,MANAGER.TECH
#S9	EXEC		9	9	WED 9:16A	TOM,MANAGER.VESOFTD

```
5 JOBS:
```

```
0 INTRO
0 WAIT; INCL 0 DEFERRED
5 EXEC; INCL 4 SESSIONS
0 SUSP
```

```
JOBFENCE= 6; JLIMIT= 2; SLIMIT= 40
```

JOBNUM	STATE	R	SCHED-CONDITION	SCHEDULED-INTRO	JOB NAME
#A1	SCHED	+		SMTWRFA 0:10	GETDFREE,MANAGER.TECH
#A2	SCHED	+		-MTWRF- 0:15	DAILY,MGR.XPRESS
#A3	SCHED	+		S-----A 0:35	DISCLEAN,MANAGER.SYS
#A4	SCHED	+		-MTWRF- 1:30	BACKUP,MANAGER.SYS
#A5	SCHED	+	WHENEVER BETWEEN(HPDAY,2,6) AND...		REMOTE,USER.XPRESS
#A6	SCHED	+	WHENEVER (HPDATE=1)		REPORT,MANAGER.SYS
#A7	SCHED		FRI	9/16/94 10:00	TESTSCHD,MANAGER.SYS

```
7 STREAMX SCHEDULED JOBS.
```

The MPE :SHOWJOB command has been enhanced to display STREAMX scheduling information as well as MPE :SHOWJOB information. When appropriate, STREAMX scheduling information is automatically displayed after the status section of the MPE :SHOWJOB command. In addition, we have created a new %SHOWJOB userset of @A to represent all STREAMX scheduled jobs.

The SCHED-CONDITION/SCHEDULED-INTRO columns display different information depending upon whether or not the job repeats on specific days, is scheduled to submit on a particular day and time or if the job should be launched when a particular condition occurs. Repeating jobs are indicated by a "+" character after the word "SCHED". For jobs that are scheduled for a particular day and time, that information is displayed much the same as MPE scheduled jobs.

For conditional jobs, as much of the condition that can be displayed on one line will be printed, followed by "..." if the conditional expression is longer.

This is essentially the same format as the %SHOWJOB command, and shows the same information as the %SEC SHOWSCHED command. %SEC SHOWSCHED, however, will display the entire condition under which a job will be submitted.

Since the default is to display both jobs and sessions, simply typing %SHOWJOB alone will display MPE jobs and sessions, MPE scheduled jobs, the MPE status block, and STREAMX scheduled jobs in that order. To display only STREAMX scheduling information, type %SHOWJOB JOB=@A. To suppress STREAMX scheduling information, include ;NOSEC as part of the command.

%SHOWLVAR

Syntax: `%SHOWLVAR [varset] [,...]`

Examples: `%SHOWLVAR COUNTER`
`%SHOWLVAR J, K, L`
`%SHOWLVAR K#, FOO@`

The %SHOWLVAR command shows local variables. Please see "[Local variables](#)" in the MPEX Variables chapter of this manual for more details.

%SHOWME

Syntax: `%SHOWME [;DETAIL]`

Examples: `%SHOWME;DETAIL`

The MPE :SHOWME command has been enhanced to display additional information when you have changed from your original LOGON to a new LOGON using the %CHLOGON or %CHGROUP commands. The example below shows the additional information that is displayed when the optional ;DETAIL parameter is used.

```
%chlogon tom,vanilla.user,tom
Welcome! You are now signed on.
%showme;detail
CURRENT LOGON: TOM,VANILLA.USER,TOM
CAPS: ND,SF,BA,IA
ORIGINAL LOGON: TOM,MANAGER.SYS,PUB

USER: #S135,VANILLA.USER,TOM      (IN PROGRAM)
RELEASE: B.40.00   MPE/iX HP31900 B.30.45   USER VERSION: B.40.00
CURRENT: THU, JUN 17, 1993, 11:35 AM
LOGON:   THU, JUN 17, 1993, 9:02 AM
CPU SECONDS: 27          CONNECT MINUTES: 153
$STDIN LDEV: 11          $STDLIST LDEV: 11
```

%SHOWOUT

Syntax: `%SHOWOUT spoolfileset [; listfile]`

Examples:

```
%SHOWOUT $STDLIST.@.@
%SHOWOUT @.@.PROD(SPOOL.JSNAME="COMP010")
%SHOWOUT @.@.@(SPOOL.OUTPUTPRI>=5)
```

The %SHOWOUT command shows information on output spool files. In this, it's similar in purpose to MPE's :SHOWOUT, but is appreciably more powerful:

- It lets you select which spool files you want to list; for instance, to find the PROD account spool files belonging to the job with job name COMP010, you can just say

```
%SHOWOUT @.@.PROD(SPOOL.JSNAME="COMP010")
```

Similarly, to find the spool files of all jobs that terminated with an error, you may enter:

```
%SHOWOUT @.@.@(SPOOL.JOBABORTED)
```

- Its output has rather more interesting information:

```
SPFile#  Filename  Pri  JOB#:jobname,user.acct,group  Ready Date
#0869    LP        7   #J493:SortCity,MGR.SALES,PUB  WED 11:45AM
#0733    CONTRACT  6   #J427:CONT,TEST.AP,DATA      TUE 11:23AM
#01181   $STDLIST  3   #J620:COMP010,JACK.AP,DEV    FRI  9:24AM
#01198   $STDLIST  3   #J632:COMP030,PETE.AP,DEV    FRI  9:55AM
ERR!
```

The output includes:

- The full logon of the job involved (including the session name).
- The spool file's output priority.
- The date that the spool file was made ready. If the date is within the past week, it's shown as a day of week (e.g. WED); if it's earlier, it's shown as the day of month and month name (e.g. 17APR). If the spool file has not yet been made ready, "(OPENED)" is shown in this column.
- The string "ERR!" if the job is marked as having terminated abnormally (i.e. because a command that was not preceded by a !CONTINUE got a fatal error) — the SPOOL.JOBABORTED selection condition lets you select by this flag.

We think that the %SHOWOUT command is a good replacement for :SHOWOUT — just the fact that it shows the job name can make it worthwhile.

MPE/iX Native Mode Spooler note

The Native Mode Spooler on MPE/iX does not retain the spoolfile's group name. If you specify a group name in %SHOWOUT on MPE/iX, it will be effectively ignored. For example:

```
%SHOWOUT $STDLIST.PUB.PROD...
```

is the same as

```
%SHOWOUT $STDLIST.@.PROD...
```

%SHOWOUTJ

Syntax:

```
%SHOWOUTJ [jobsessionname] [,user.account]
```

Examples:

```
%SHOWOUTJ DAILY,@.VESOFT
%SHOWOUTJ XXX           is the same as %SHOWOUTJ XXX,@.@
%SHOWOUTJ ,USER.ACCT    is the same as %SHOWOUTJ @,USER.ACCT
```

%SHOWOUTJ is simply a very useful special case of %SHOWOUT. It shows you all the spool files belonging to jobs/sessions with the given job/session name, user, and account.

For example, to see all the spool files belonging to jobs/sessions named DAILY that logged on as any user in the VESOFT account:

```
%SHOWOUTJ DAILY,@.VESOFT

SPFile#  Filename  Pri  JOB#:jobname,user.acct,group  Ready  Date
#0923    LP         7  #J493:DAILY,MANAGER.VESOFT,PUB  TUE 12:23AM
#0755    $STDLIST   1  #J493:DAILY,MANAGER.VESOFT,PUB  TUE 12:23AM
#01150   $STDLIST   1  #J505:DAILY,TEST.VESOFT,PUB     (OPENED)
```

This command is quite useful if you frequently have many jobs that log on with the same user ID but with different job names.

As we mentioned above, this command is just a special case of the much more powerful %SHOWOUT command. Saying

```
%SHOWOUTJ X@,Y@.Z@
```

is the same as saying

```
%SHOWOUT @.@.Z@ (SPOOL.JSNAME MATCHES "X@" and &
                  SPOOL.USER MATCHES "Y@")
```

%SHOWPAUSED

Syntax:

```
%SHOWPAUSED
```

%SHOWPAUSED shows information about jobs that were suspended via the %PAUSEJOB command. For more information, see the documentation on [%PAUSEJOB](#).

%SHOWPROC

Syntax:

```
%SHOWPROC [[PIN=][#P]pin | [PIN=](pin [,...])]
           [[JOB=]userset | JOB=(jobID [,...])]
           [;FORMAT=[{SUMMARY|FILES}]]
           [;FILES=filesset]
           [;[NO]TREE]
           [;{USER|ANYUSER}]
           [;SYSTEM]
           [;C[HANGE]]
           [;[NO]TRUNC]
```

Examples:

```
%SHOWPROC #J232
%SHOWPROC (190,56,110,108)      (* list of pins *)
%SHOWPROC JOB=(#J132, #J134)    (* list of jobs *)
%SHOWPROC #J232; FORMAT=FILES  (* info on open files *)
%SHOWPROC BATCH; FILES=@.DB.PROD(ISPRIV)
%SHOWPROC LDEV=MODEM
%SHOWPROC LDEV=MODEM;CHANGE
```

MPEX's %SHOWPROC command is an improved version of the MPE/iX :SHOWPROC command; it provides the following enhancements:

- it displays the job/session number, job/session name, user name, account name, and group name before the first PIN of each job/session;
- it lets you list just those processes which have used CPU since the last %SHOWPROC command you entered;
- it will show you information about any (or particular) files that the processes you're displaying have open (perm and temp files);
- it shows you the command line or ;INFO= string of each process (which may be truncated using the ;TRUNC keyword if the process nesting is too deep);

%SHOWPROC entered without parameters will show all processes in your current job/session:

```
%SHOWPROC
QPRI  CPUTIME   STATE  JOBNUM  PIN  (PROGRAM) STEP
      #s187, eugene,manager.tech,eugene (ldev 20)
C152  0:02.425 WAIT   S187    17   :RUN MPEX.PUB.VESOFT
C152  0:00.304 WAIT   S187   105 (MPEX.PUB.VESOFT)
C158  0:11.023 READY  S187    88   (MAIN.PUB.VESOFT)
```

You may specify a PIN or list of PINs to show particular processes, or a job/session ID, a list of job/session IDs or a VESOFT userset to show all processes for a particular job or or set of jobs (in this discussion, job means job or session). If you don't specify any job, userset or PIN, you will be shown information about your current job/session.

"Normal" users (users without SM or OP capability) will only be allowed to view information on processes logged on with their same user.account. SM/OP users by default will be shown information about any jobs/sessions that match the userset they specify. They can limit the output to jobs/sessions matching their user.account by saying ";USER". (This is mostly for compatibility with MPE/iX.)

If you have OP capability, you can show any user process(es), but not system processes. If you have SM capability, you can show any user or system process(es).

If you specify a PIN (or list of PINs), %SHOWPROC defaults to ;NOTREE – only show the process(es) whose PIN(s) were specified, not their sons.

If you specify a jobID (or list of jobIDs) or VESOFT userset, the default is ;TREE — all processes in each qualifying job are shown.

The ;CHANGE keyword allows you to show just those processes that have used CPU time since the last execution of %SHOWPROC! (You must do at least one %SHOWPROC, without ;CHANGE, that includes the process(es) you are interested in before using ;CHANGE.)

For example, if you do a

```
%SHOWPROC OPERATOR.SYS
```

and then later do a

```
%SHOWPROC OPERATOR.SYS; CHANGE
```

the second %SHOWPROC will only show those processes that have used CPU time since the execution of the first %SHOWPROC; the CPUTIME column will show the CPU used since the last %SHOWPROC rather than (as it normally does) the total CPU used by each process.

The output of the %SHOWPROC command includes the command line or INFO string passed to each program. When a process tree has several nested processes or the INFO strings are quite long, the ;INFO= that we display will wrap around to the next line of your screen. While this isn't serious, it can make reading the output of this command difficult, so we have created a ;TRUNC keyword that will truncate the output at 80 characters. You may want to set this as a default using the VESOFTDEFAULTSHOWPROC variable. If you have set this variable, you can use ;NOTRUNC to override the VESOFTDEFAULTSHOWPROC setting. Truncated lines are identified with a '\$' character in the last position.

Showing which files a process has open (;FORMAT=FILES)

MPEX's %LISTF..., ACCESS (documented [elsewhere](#) in this manual) has long been one of our most popular commands. It gives you information about everyone who is accessing a particular file (or filesset). But, sometimes what you want is just the opposite: you want to know which files a particular user (or userset) is accessing. The ;FORMAT=FILES option gives you this information!

MPEX COMMAND REFERENCE: %SHOWSAVED

For example, if you want to see what the job `LONGTIME,BATCH.PROD` is doing at the moment, you could type:

```
%SHOWPROC LONGTIME,BATCH.PROD; FORMAT=FILES
```

This will show you, for every file the job currently has open (in addition to the normal process information %SHOWPROC displays):

- the file number, full name, file code and record size;
- the type of access the file is opened for (Read, Write, Append, execute, lock);
- the current EOF;
- the current record number;
- the number of disk I/Os this process has performed against this file.

%SHOWPROC shows this information for both temporary and permanent files!

The `;FORMAT=FILES` keyword shows all of the files that every process in the process tree has open. Often, this provides too much information — you may be interested in only one particular file or fileset instead of every file. Even though `%LISTF...,ACCESS` shows the information for a file or fileset, it also shows you every user accessing the file, which again can be "too much" information (besides, %SHOWPROC shows more information about how this process is accessing the file than %LISTF does). To limit the output of the %SHOWPROC command to only those files that are in a given fileset, use the keyword `;FILES=fileset` instead (in fact, `;FORMAT=FILES` is equivalent to `;FILES=@. @. @`).

%SHOWSAVED

Syntax:

```
%SHOWSAVED;FILE=savefile
```

Examples:

```
%SHOWSAVED;FILE=SAVEDJBS
```

%SHOWSAVED shows information about jobs that were saved via the %SAVEJOB command. For more information, see the documentation on [%SAVEJOB](#).

%SHOWTREE

Syntax:

```
%SHOWTREE
```

%SHOWTREE shows the entire process tree of your session, e.g.

```
%SHOWTREE
24      C.I.
    19    MPEX.PUB.VESOFT
        28    MAIN.PUB.VESOFT (active) [you are here]
        53    EDITOR.PUB.VESOFT
```

Note: Since MPEX is a CM shell program that runs the NM MAIN program (on MPE/iX systems), the "[you are here]" marker will appear next to the MAIN program and not the MPEX program.

The "[you are here]" marks the MPEX process; the indentation indicates who is whose son (PIN 53 is the son of MAIN [PIN 28]).

Once you know that, say, SPOOK5.PUB.SYS is PIN 17, you can type:

```
%RUN 17
```

to re-activate it or

```
%KILL 17
```

to kill it. Of course, you could have always said

```
%SPOOK5
```

or

```
%KILL SPOOK5.PUB.SYS
```

(referred to the son process by its program name) — but referring to it by PIN can also be useful sometimes (to save typing or to resolve ambiguities if you have two son processes with the same name).

%SHOWVAR

Syntax:

```
%SHOWVAR [varset] [,...]
```

Examples:

```
%SHOWVAR          << shows all your own variables >>
%SHOWVAR @         << shows all variables, including >>
                   << system-defined variables >>
%SHOWVAR X, TEMPV@
```

The %SHOWVAR command shows the values of the specified MPEX variables (see the [MPEX variables](#) chapter for more information).

Typing

```
%SHOWVAR
```

will show all of the modifiable variables (mostly the ones that you've set yourself and the ones that MPEX uses for control purposes).

On the other hand,

```
%SHOWVAR @
```

will show all the variables, including the predefined non-modifiable ones, like HPACCOUNT, HPUSER, etc.;

MPEX COMMAND REFERENCE: %SPOONFEED

for instance:

```
AP = PURGERBM SEGMENT,EXP'  
MPEXPREFIX = !  
MPEXFILESET = MY@  
VESOFTPAGESIZE = 60  
HPACCOUNT = VESOFT  
HPCMDNUM = 6  
HPCPUNAME = SERIES MICRO/XE  
HPDATE = 30  
HPDATEF = SAT, JAN 30, 1991  
...
```

To see the value of a specific variable or set of variables, you can say

```
%SHOWVAR AP  
AP = PURGERBM SEGMENT,EXP'
```

or

```
%SHOWVAR HPDATE@,A?  
HPDATE = 30  
HPDATEF = SAT, JAN 30, 1991  
AP = PURGERBM SEGMENT,EXP'
```

As you see, you can use wildcards (@, ? and #) to select those variables you want.

%SPOONFEED

Syntax:

```
%SPOONFEED progrname, cmd  
%>cmd      (to execute cmd as a SPOOK5 command)  
%'cmd      (to execute cmd as a QUERY command)
```

Examples:

```
%>TEXT 1234  
%>LIST 10/20  
%'FIND CUST-NO="X985"  
%'REPL CUST-NAME="JOHN SMITH & CO."; END
```

Many programs (EDITOR, QUERY, etc.) allow you to execute some MPE commands from within them simply by prefixing the command with a ":". Similarly, all MPEX HOOKed utilities let you execute MPEX commands from within them by prefixing the command with a "%".

Imagine that you could similarly execute QUERY commands or commands of any other utilities — instead of running QUERY, executing the command, exiting QUERY, and then re-running it the next time you need to do a QUERY command, you could just say

```
%'FIND STATE=CA,TX,WA
```

All that you have to do to execute a QUERY command is to prefix it with a "'" character.

```
%>TEXT 1234
%>LIST 1/20
```

You might even put all these ">"-prefixed commands in a command file; the same command file can mix MPEX commands, QUERY commands, and so on!

We call this process "spoon-feeding" — we keep QUERY or whatever else as a son process and pass it one command at a time. As soon as the program involved executes that command and prompts for more input, control is returned to MPEX (or whatever MPEX HOOKed program you're using MPEX from). This actually works regardless of what sort of input the "spoon-fed" program prompts you for (a command or some other kind of input).

Thus, a typical "spoon-fed QUERY" conversation might look like this:

```
%' BASE=MYDB
HP32216D.00.08 QUERY/3000

PASSWORD =                << QUERY prompt >>
%' MYPASS
MODE =                    << QUERY prompt >>
%' 5
>                          << QUERY prompt >>
%' FIND CUST-N0="X987"
...

```

Every time QUERY prompts you for input, control is returned to MPEX, and QUERY remains suspended until you enter the next command that starts with "'" — this command will then re-activate QUERY and send it the data that you want.

Note that this sort of approach might take some getting used to. For instance, many programs (like QUERY and EDITOR) will output their own prompt to you before re-activating MPEX; thus, you'll see not just MPEX's "%" prompt but also the program's prompt before it. Also, in the example above, QUERY turned off echo when prompting for the password; echo will remain off for MPEX until you either do a "%SET ECHO=ON" command or send the password to QUERY (which will cause QUERY to turn echo back on). However, all in all, we think that this can be a very powerful time-saving feature.

Another way to look at spoon-feeding is that it doesn't "mode you in"; instead of having "EDITOR mode", "MPE mode", "QUERY mode", etc., you can readily execute any kind of command you want from one environment (typically either MPEX or your favorite HOOKed text editor).

How to use spoonfeeding

All of the above has been an introduction to the benefits of what we believe to be a rather novel feature. How exactly can you get it to work for you?

Well, technically speaking, the fundamental spoon-feeding command is

```
%SPOONFEED progname, cmd
```

MPEX COMMAND REFERENCE: %SPOONFEED

For instance, to execute a QUERY command, you'd say:

```
%SPOONFEED QUERY, FIND CUST-NO="X987"
```

This will execute QUERY.PUB.VESOFT (this program must be an MPEX HOOKed program in PUB.VESOFT — QUERY is HOOKed automatically when you install our software) and pass to it the FIND command.

Of course, if you had to type "SPOONFEED QUERY, " before every QUERY command you wanted to execute, it wouldn't be much better than having to run QUERY and execute the command directly from QUERY.

Fortunately, we let you define the meaning of any special character (except for the reserved characters "%", "!", "?", "\$", ":" and ", " [comma]) when used as the first character of an MPEX command.

In the default MPEXMGR.PUB.VESOFT file, we say

```
:%SETVAR MPEXPREFIXDEF39 "SPOONFEED QUERY,"
```

This means: "Set the DEFINITION of PREFIX character 39 (which is the ASCII code for a single quote) to be 'SPOONFEED QUERY,'". (If you haven't memorized the ASCII table yet, you could have said %CALC ORD("'") to find out the ASCII code.) Whenever MPEX sees a command that starts with a "'", it will replace the quote by "SPOONFEED QUERY,", which will then cause the command to be executed as a QUERY command.

Needless to say, you can in this way define your own spoon-fed programs. If, for instance, you frequently use KSAMUTIL, you can say:

```
%HOOK KSAMUTIL.PUB.SYS
```

to make an MPEX HOOKed copy of it in PUB.VESOFT, and then add a line to MPEXMGR.PUB.VESOFT that says

```
:%SETVAR MPEXPREFIXDEF94 "SPOONFEED KSAMUTIL,"
```

This will define "^" (ASCII 94) as the prefix character for KSAMUTIL command — of course, you can use some other character if you prefer, or you might even set up a command file called "K" that says

```
ANYPARM CMD  
SPOONFEED KSAMUTIL,!CMD
```

and then you'll be able to execute KSAMUTIL commands from MPEX simply by saying

```
%K BUILD MYFILE;KEYFILE=MYFILEK;KEY=...
```

(There's nothing that says that you have to use special characters to identify spoon-feeding commands.)

An important warning

As we mentioned before, if you use spoon-feeding you might have to get used to a few new considerations. One of the most important ones has to do with the fact that the spoon-fed process remains alive even when you're not in it. In a way, this is obvious, since this is the whole point of spoon-feeding; however, this also means that any files that the process has opened remain opened and any resources that it has locked remain locked.

For example, a problem can happen if you're spoon-feeding QUERY and do a `% FIND` command without previously doing an `% ASSIGN LOCKOPTION=OFF`. The `FIND` command will lock the entries it finds, which is not bad by itself; however, if you then run your own program from within MPEX and it tries to lock something, the program will either get an error indicating that it tried to lock something while the session already holds a lock.

Furthermore, if your program has `CAP=MR` (which explicitly turns off this lock-time checking), you can cause a mini-deadlock — your program will be waiting for a lock that is held by QUERY, while QUERY will be waiting for re-activation by MPEX, which in turn is waiting to be re-activated by your program. You would then have to do a `[BREAK]` and `:ABORT` to get out of this.

Getting information on which programs you're spoonfeeding

Here are a few ways in which you can see the state of the programs you are currently spoon-feeding:

- A `%SHOWTREE` will show you all of your son processes, including the ones you're spoon-feeding.
- The PIN (Process Identification Number) of a spoon-fed process is kept in the JCW `MPEXSONEXECPIN $\textit{progrname}$` , where the " $\textit{progrname}$ " is fully qualified. Since the "." character isn't legal in a variable name, the "."'s are changed to the letter "Z". For instance, the PIN of a spoon-fed QUERY would be in `MPEXSONEXECPINQUERYZPUBZSYS`.
- If you've forgotten where you are in a particular spoon-fed process, remember that all spoon-fed processes are running MPEX HOOKed programs — you can use the MPEX HOOK `LISTREDO` command to see which commands you've executed through this spoon-fed program. For example, entering `"':LISTREDO"` (or `' , ,`) will show you the last 20 commands you've spoon-fed to QUERY.

%SPOONINIT

Syntax:

```
%SPOONINIT programfile {;MPEX %run-parm} [;...]
```

Examples:

```
%SPOONINIT QEDIT;PARAM=7
```

Occasionally, you may need to supply a "runtime" parameter to the program being spoonfed (such as a `;PARAM=` value or an `;INFO=` string). `%SPOONINIT` allows you to do this; however, it must be used as the FIRST "spoonfeed" type command for the program and does not allow you to actually specify a command to be "spoonfed".

MPEX COMMAND REFERENCE: %SUBMIT

Although ;INFO= and ;PARM= are most likely to be the only parameters you would need to specify, the following parameters may be used as well (of course, you should not use any parameters from this list if they cause your program to abort outside of MPEX):

```
;DL
;INFO
;LMAP
;MAXDATA
;NMHEAP      (NM)
;NMSTACK     (NM)
;PARM
;PRI
;STACK
;STDLIST
```

The following %:RUN parameters should be avoided as they will interfere with our processing of the spoonfeed command:

```
;DEBUG (used for debugging, we cannot supply input in this case)
;GOON
;INMSG
;INPUT
;KILL
;NEW (we supply this in our command)
;NOACTIVATE (we supply this in our command)
;NOPRIV
;STDIN (SPOONFEED provides the STDIN input)
;UNSAT (used for debugging)
;XL= (NM) or ;LIB= (CM) (we supply this in our command)
```

%SUBMIT

Syntax:

```
%SUBMIT "[joblogon] [;jobparms]", mplexcommand
```

Examples:

```
%SUBMIT ";OUTCLASS=,1;INPRI=2;PRI=ES", &
        COBOL AP@.SOURCE, AP@.PUB
%SUBMIT "MANAGER.SYS", PURGE LOG####.PUB.SYS
%SUBMIT "MGR.PROD;OUTCLASS=SLOWLP,3", RUN MYPROG;PARM=1234
```

The %SUBMIT command makes MPEX submit a job that will execute the given MPEX command.

- You may (optionally) indicate that the job is to be submitted under somebody else's user ID or with some special job parameters.
- You may execute either an MPEX command or a normal MPE command (RUN, STORE, a UDC, etc.); if the MPE command has the same name as an MPEX command (like FCOPY), precede it with a ":".

- All the file equations that are in effect in your session, and the current value of your HPPATH variable, will automatically be copied to the created job.
- You may indicate who MPEX should notify of the job's completion by setting the MPEXTELLBACK variable.
- You may indicate how MPEX should inform you of the job's completion by setting the MPEXTELLBACKCMD variable.

(See the [%SCHEDULE](#) chapter and "Batch execution ('\$' prefix)" in the MPEX Commands chapter for more information.)

To execute an MPEX command in a job stream, you can simply say

```
%$COBOL AP@.SOURCE, AP@.PUB
```

The "\$" prefix tells MPEX to perform the command offline — MPEX will automatically generate a job stream that executes the above command.

It'll also copy all the file equations (such as :FILE COPYLIB=...) that you may have in your session into the job, so that the job environment is as close as possible to your session environment.

The %SUBMIT command simply extends this capability to allow you to specify a different logon ID and :JOB card parameters for the job stream. Saying

```
%$COBOL AP@.SOURCE, AP@.PUB
```

will execute "COBOL AP@.SOURCE, AP@.PUB"

- under your own logon id
- with the job parameters ";OUTCLASS=,1".

What if you want to submit it at input priority 2 with CPU priority ES? Just say

```
%SUBMIT ";OUTCLASS=,1;INPRI=2;PRI=ES", COBOL AP@.SOURCE, AP@.PUB
```

If you say

```
%SUBMIT "MGR.PROD;OUTCLASS=SLOWLP,3", COBOL AP@.SOURCE, AP@.PUB
```

then it'll not only use the given job card parameters but also the given logon ID (MGR.PROD).

Saying just

```
%SUBMIT "MANAGER.SYS", PURGE LOG####.PUB.SYS
```

will only change the logon ID (MANAGER.SYS) while keeping the default job card parameters (;OUTCLASS=,1).

Of course, if you try to submit something under somebody else's logon ID, MPEX will prompt you for the passwords (unless you have SM capability or AM capability in your own account).

If you always want to submit your jobs with a particular set of !JOB card parameters (other than the default ";OUTCLASS=,1"), you might want to add a %SETVAR MPEXDEFAULTJOBPARMS

MPEX COMMAND REFERENCE: %SYNTAX

to your MPEXMGR, MPEXMGR.PUB or MPEXMGR.PUB.VESOFT file to set the special MPEXDEFAULTJOBPARMS variable to your default !JOB card parameters.

For example, you might say

```
%SETVAR MPEXDEFAULTJOBPARMS "OUTCLASS=,2;PRI=ES"
```

If you do this, all the jobs you submit with the "\$*command*" syntax (or with the %SUBMIT command in which you only specify a different logon ID, not a different set of !JOB card parameters) will be submitted with OUTCLASS=,2 and PRI=ES.

%SYNTAX

Syntax:

```
%SYNTAX [mpexcommand]
```

Examples:

```
%SYNTAX ALTFIL  
%SYNTAX PRINT
```

MPEX has a lot of new commands, and adds many features to existing MPE commands. To help you take full advantage of the increased power MPEX gives you over your HP3000, we provide a very easy-to-use conceptual online help facility (see the documentation on the [%HELP command](#) for details).

Sometimes, however, all you want is a quick reminder of the syntax of a particular command. For this purpose, we provide the %SYNTAX command.

Typing

```
%SYNTAX ALTFIL
```

for example, will take you directly to the syntax diagram for MPEX's ALTFIL command.

Typing

```
%HELP ALTFIL
```

would give you a list of topics related to the %ALTFIL command, and ask you which one you would like help on.

%TELL

Syntax:

```
%TELL userset [[;] text]
```

Examples:

```
%TELL EUGENE,@.&&ONLINE Lunch time!  
%TELL @,MANAGER.@@,MGR.@ System will be down tonight...  
%TELL LDEV=113 Is anyone there?  
%TELL LDEV=DIALIN Please hang up soon or be aborted.
```

The %TELL command is an enhanced version of MPE's :TELL command. It allows you to specify which sessions to :TELL to using VESOFT usersets (see the Usersets section of the SECURITY User Manual).

%TRAPERROR

Syntax:

```
%TRAPERROR
%command
...
%command
%{IFERROR|CLEANUP}
%    command
...
%    command
%{ENDIFERROR|ENDCLEANUP}
```

Although MPE lets you do rudimentary error handling with :CONTINUE, :IF/:ELSE/:ELSEIF/ :ENDIF and the CIERROR JCW, this can prove to be very cumbersome.

For instance, say that you want to do the commands A, B, C, and D, but once one of them gets an error, you want to avoid doing the others without aborting the UDC, command file or job (perhaps because you want to do some cleanup at the end).

You'd have to say something like:

```
SETJCW CIERROR=0
CONTINUE
A
IF CIERROR=0 THEN
  CONTINUE
  B
  IF CIERROR=0 THEN
    CONTINUE
    C
    IF CIERROR=0 THEN
      CONTINUE
      D
    ENDIF
  ENDIF
ENDIF
ENDIF
IF CIERROR<>0 THEN
  TELLOP ERROR IN THE PROCESSING
ENDIF
```

MPEX has a very useful construct that can let you do this much more easily:

```
TRAPERROR
A
B
C
D
IFERROR
    TELLOP ERROR IN THE PROCESSING
ENDIFERROR
```

This syntax tells MPEX to do the commands between the %TRAPERROR and the %IFERROR until an error occurs, and then, if the error occurs, do the commands between the %IFERROR and %ENDIFERROR (and then continue with the rest of the UDC, command file or job).

This can be very useful. For instance, you can make all your production jobs look like:

```
:JOB...
:COMMENT    The first thing we do is :RUN CI.PUB.VESOFT, which
:COMMENT    provides a very good emulation of MPE (plus all the
:COMMENT    new features.)
:RUN CI.PUB.VESOFT
TRAPERROR
<< all the commands you'd normally put into your job >>
IFERROR
    TELLOP !HPJOBNAME FAILED, ERROR !HPCIERMSG
ENDIFERROR
EXIT
:EOJ
```

The jobs will function the same way as before (subject to the relatively few minor incompatibilities between MPEX and MPE — test before you commit to this!), but instead of just blindly aborting, they catch the error and do whatever cleanup is necessary — in this case, just a :TELLOP, but it could also include any necessary :PURGEs and such.

%CLEANUP

Another form of error handling that we've often found useful is doing several commands, and then, regardless of whether an error occurred or not, doing some "cleanup" commands, e.g.:

```
%BUILD F1
%BUILD F2
%TRAPERROR
%   RUN PROG1
%   RUN PROG2
%   ...
%CLEANUP
%   :PURGE F1
%   :PURGE F2
%ENDCLEANUP
```

Here, the commands between the TRAPERROR and the CLEANUP are executed, and then, whether or not an error occurred, the commands between the CLEANUP and ENDCLEANUP are executed. Then and only then — after the cleanup commands are done — if an error occurred in the TRAPERROR/CLEANUP block, the rest of the UDC, command file or job is flushed (subject, of course, to any other TRAPERROR constructs in which it is itself nested).

If you want to immediately "jump" to the IFERROR or CLEANUP block you can use:

```
%ESCAPE errnum
```

This sets HPCIERR to *errnum*, sets CIERROR to the absolute value of *errnum*, and skips all statements until the next IFERROR or CLEANUP.

For example:

```
...
TRAPERROR
IF VEFINFO("FILE1").EOF//VEFINFO("FILE1").FLIMIT > .80 THEN
    ESCAPE 1
ELSEIF VEFINFO("FILE2").EOF//VEFINFO("FILE2").FLIMIT > .80 THEN
    ESCAPE 2
ELSEIF VEFINFO("FILE3").EOF//VEFINFO("FILE3").FLIMIT > .80 THEN
    ESCAPE 3
ENDIF
...
IFERROR
    ECHO ERROR:
    ECHO   FILE!HPCIERR is more than 80% full!!!
    ECHO   Terminating job submission.
    EXIT
ENDIFERROR
```

For additional information, see the [%ESCAPE command](#) documented earlier in this manual.

%UNALIAS

Syntax:

```
%UNALIAS aliasname
```

Examples:

```
%UNALIAS WG
%UNALIAS SS
```

%UNALIAS and %ALIAS are closely related commands; please see the documentation on [%ALIAS](#).

%VEAUDIT

Syntax: `%VEAUDIT command`

Examples: `%VEAUDIT LISTUSER @.@`
`%VEAUDIT ALTUSER CAP=SM-MANAGER.SYS;CAP=-SM`

The %VEAUDIT prefix allows VEAUDIT users to enter VEAUDIT accounting structure maintenance commands within MPEX. Please see "Accounting Structure Maintenance" in the VEAUDIT User Manual for details.

%VECMDCH

Syntax: `%VECMDCH fileset [;REMOVE]`

Examples: `%VECMDCH @.PUB.PROD(ISPROG)`

This command allows you to change a program fileset so that it uses STREAMX instead of STREAM (i.e. it replaces calls to the **COMMAND** intrinsic with calls to the **VECMMD** intrinsic).

To change a fileset from **VECMMD** back to **COMMAND**, use the ";REMOVE" option.

Please see the What if your program does a :STREAM programmatically chapter in the SECURITY User Manual.

%VEOPENCH

Syntax: `%VEOPENCH fileset [;REMOVE]`

Examples: `%VEOPENCH @.PUB.PROD(ISPROG)`

This command allows you to change a program fileset so that it uses VEOPEN instead of DBOPEN (i.e. it replaces calls to the **DBOPEN** intrinsic with calls to the **VEOPEN** intrinsic).

To change a fileset from **VEOPEN** back to **DBOPEN**, use the ";REMOVE" option.

Please see the Making Program Which Source Code You Don't Have Call VEOPEN chapter in the SECURITY User Manual.

%WARN

Syntax: `%WARN userset [[;]text]`

Examples: `%WARN ONLINE System going down soon; please log off.`
`%WARN LDEV=113 Last chance to log off gracefully...`
`%WARN LDEV=DIALIN Please hang up soon or be aborted.`

The %WARN command is an enhanced version of MPE's :WARN command. It allows you to specify which sessions to :WARN using VESoft usersets (see the Usersets section of the SECURITY User Manual).

Note: Just as in MPE, the WARN command can only be issued from the console, unless permitted via the ALLOW command.

%WARNF

Syntax: `%WARNF userset [[;]text]`

Examples: `%WARNF ONLINE System going down soon; please log off.`
`%WARNF @,MANAGER. @+@,MGR. @ System will be down tonight...`
`%WARNF LDEV=113 Last chance to log off gracefully...`
`%WARNF LDEV=DIALIN Please hang up soon or be aborted.`
`%WARNF ACCESSING=CUST.DB.PROD Please exit CUSTDB now...`

%WARNF is just like %WARN except that it displays the message right over the function keys (on most new HP terminals), rather than on the actual display portion of the screen. This makes it easy for you to send messages to people without overwriting any of the data that they might have on their screen. To erase the message, users can simply press [RETURN].

Unlike the :TELL (or %TELL) command, %WARNF does not include any information about who sent the message. To avoid having to manually include your name (or session number) in every %WARNF message you send, MPEX will prefix the message with the contents of the variable MPEXWARNFPREFIX (if you %SETVAR it).

For example, to prefix all %WARNF messages with "From Ron: ", you would do a

```
%SETVAR MPEXWARNFPREFIX "From Ron: "
```

before you do any %WARNFs. In fact, you may want to include this %SETVAR in your MPEXMGR file (see "[Specifying commands to be executed each time MPEX is entered](#)" in the Running MPEX section of this manual).

%WARNF will not work on non-HP terminals or on really old HP terminals (like the 2621 or the 264x series) — it will either display the message on the normal part of the screen, with some garbage characters before it or might (for non-HP terminals) trigger some strange terminal-specific behavior. Be careful.

%WHEREIS

Syntax: `%WHEREIS command`

Examples:

```
%WHEREIS EDITOR
----- EDITOR      MPE command
          EDITOR      Program file in PUB.SYS

%WHEREIS MPEX@
----- MPEX        System-level UDC in MPEXUDC.PUB.VESOFT
          MPEX        Program file in PUB.VESOFT

----- MPEXHELP     MPEX command

----- MPEXLDIR     Command file in PUB.SYS

----- MPEXMGR      Command file in WORK.VESOFT
          MPEXMGR      Command file in PUB.VESOFT

----- MPEXSTOR     MPEX command
```

This command takes the name of a command as its parameter and displays how MPEX would interpret the name if it were issued as a command.

WHEREIS uses the same rules that MPEX uses at execution time to determine how a command is executed. The command identifies all instances of the command that exist, the first one shown is the one that would actually be executed.

The command parameter accepts wildcards, as shown in the second example above.

The command is useful if MPEX appears to execute a different command from the one you intended. It shows you if you have conflicts between the names of your UDCs and command files, and also if your HPPATH gives you access to the commands that you expect it to.

MPEX uses the following precedence when determining which command to execute:

- 1 MPEX aliases.
- 2 User-level UDCs.
- 3 Account-level UDCs.
- 4 System-level UDCs.
- 5 MPEX commands.
- 6 MPE commands.
- 7 Temporary programs or command files in your HPPATH.
- 8 Permanent programs or command files in your HPPATH.

If the command parameter includes wildcard characters, all matching commands will be displayed. However, MPE commands are not shown when wildcard characters are used.

%WHILE...%ENDWHILE

Syntax:

```
%WHILE logicalexpression [DO]
%> command1
%> command2
...
%> commandN
%ENDWHILE
```

Examples:

```
%SETVAR I 0
%WHILE I<10
%>BUILD TEST!I
%>SETVAR I I+1
%>ENDWHILE
```

The %WHILE loop executes a sequence of commands while a certain condition is true (much like a PASCAL's WHILE loop).

For instance, the above example will execute the commands

```
%>BUILD TEST!I
%>SETVAR I I+1
```

10 times, with the variable I being set to values from 0 to 9. Once the %>SETVAR sets I to 10, the condition "I<10" will no longer be true, and the %WHILE loop execution will stop.

In other words, the commands actually executed will be:

```
BUILD TEST0
SETVAR I I+1
BUILD TEST1
SETVAR I I+1
...
BUILD TEST9
SETVAR I I+1
```

When you're being prompted for %WHILE loop commands, MPEX will prompt you with "%>" instead of just "%".

%WHILE is much like MPE/iX's :WHILE command, but it gives you the full power of VESOFTE expressions (for details, see [Appendix B](#))!

Here's a sample job stream using %WHILE:

```
!JOB JOBA,MGR.PROD
!COMMENT
!COMMENT  All MGR.PROD files that have the following lines
!COMMENT  will only be allowed to run one at a time.
!COMMENT
!RUN MAIN.PUB.VESOFT;PARM=1
%WHILE MPE ("BUILD FLAGFILE; CODE=!HPJOBNUM; DISC=1")<>0 DO
%   COMMENT The following IF is in case the job that built
%   COMMENT FLAGFILE aborts before PURGEing it.
%   IF JSCOUNT("#J"+"![VEFINFO('FLAGFILE').INTCODE]")=0 THEN
%       PURGE FLAGFILE
%   ELSE
%       PAUSE 300
%   ENDIF
%ENDWHILE
%EXIT
...
!PURGE FLAGFILE
!EOJ
```

%WITHCAPS

Syntax:

```
%WITHCAPS "caplist", command
```

Examples:

```
%WITHCAPS "SM", COPY F1,F2.PUB.ACCT;YES
%WITHCAPS "OP", STORE @.@.@; *TAPE
%WITHCAPS "AM", NEWGROUP !GROUPNAME
```

Special note: Although this feature is being described in the MPEX User Manual, it is only available in MPEX command files if you are a user of both MPEX and SECURITY. It is also available within STREAMX jobs (for :: commands) and in SECURITY menus.

Often you might want to have users execute certain commands in command files, job streams or menus, that require more capabilities than they have. For instance, you might want a menu to access — in a controlled way, of course — a file that you normally wouldn't let your users access; or, you may want to set up an MPEX command file that lets users who lack SM capability build new accounts, provided that certain checks are satisfied (e.g. the new accounts don't have unusual capabilities). The WITHCAPS feature makes this possible.

Another similar SECURITY feature is the \$NOPASS mechanism for STREAMX job streams, (see "Allowing People To STREAM Certain Files Without Password Prompts" in the Batch Access Security And Job Stream Programming section of the SECURITY User Manual) which lets users stream jobs that log on as user IDs whose passwords the submitting users don't know — a user is thus allowed to perform a particular operation that requires a given capability (for WITHCAPS) or logs on as a given user ID (for \$NOPASS) without in general having that capability or being able to use the given user ID.

The command file, menu or job stream involved can then perform the proper security checks to make sure that the high-capability operation that it's trying to perform is actually legitimate.

The way the WITHCAPS mechanism works is rather simple:

- You add lines to the `STREAMX.DATA.VESOFT` file that indicate which command files/menus/job streams being executed by which users can get which capabilities.

For instance, saying

```
$WITHCAPS-PERMIT @.AP APPFILE.CMD.SYS "SM"
```

means that the file `APPFILE.CMD.SYS` can use a `WITHCAPS` command to temporarily obtain SM capability — but only while executing the particular line that contains "WITHCAPS", and only when executed by users in the AP account.

You can, of course, have multiple `$WITHCAPS-PERMITs`, but you can also restrict them by using `$WITHCAPS-FORBIDs`, e.g.

```
$WITHCAPS-PERMIT @.AP @.CMD.SYS "AM"
$WITHCAPS-FORBID CLERK.AP NEWUSER.CMD.SYS "AM"
```

which lets all AP users use all `CMD.SYS` files that acquire AM capability, but forbids `CLERK.AP` from using `NEWUSER.CMD.SYS`, a command file that would need AM capability to add a new user to the system, for instance.

Needless to say, it's very important to think out all the security ramifications of letting these users use the files which temporarily get extra capabilities. In particular, it's vital that users not have write or save access to the filesets specified in `$WITHCAPS-PERMIT`, since otherwise they'd be able to make up their own files with `WITHCAPS` commands in them.

- Once the appropriate `$WITHCAPS-PERMIT` line has been added to the `STREAMX.DATA.VESOFT` file, the job stream, menu or command file (or even `$STDIN`) would then include a

```
::WITHCAPS "caplist", cmd in a STREAMX job
```

or

```
WITHCAPS "caplist", cmd in a SECURITY menu or MPEX command file
```

For example:

```
WITHCAPS "SM", COPY F1,F2.PUB.ACCT;YES
```

will execute the `COPY` command while possessing SM (and will, of course, reset the capabilities to normal when done).

A menu option using this might look like:

```
*CAPTION ADD RECORD TO CONTROL FILE
PROMPT STRING REC = "Enter record"; &
      CHECK = (REC[0:1]<>"A" and REC[0:1]<>"D"); &
      CHECKERROR = "Not allowed to add Add or Delete records"
WITHCAPS "SM", CALC VEFWRITE ('CONTFIL,OLD;ACC=APPEND', REC)
```

Similarly, an MPEX command file (or UDC) might have a %WITHCAPS and a STREAMX job might have a ::WITHCAPS. Note that the STREAMX ::WITHCAPS will change capabilities at job submission time (for the benefit of any :: commands that you might want to do), not at job execution time (since you can already determine the job execution capabilities by submitting the job under the right user ID).

You need to keep a few things in mind when you use WITHCAPS:

- Firstly, be careful what you do when you've gotten capabilities with WITHCAPS. For instance, don't just do an operation on a file whose name the user gave you (as a parameter or an input value) — check to make sure that the user is really allowed to manipulate this file. Don't just create a new user with the parameters that the user specified; check to make sure that, for instance, the capability list doesn't include SM, AM, OP or PM.
- The WITHCAPS takes effect only for the MPEX/LOGON/STREAMX process and for processes the command passed to WITHCAPS starts up. Since we don't set the capabilities for the whole session (for security reasons), some operations (VEJOBINFO, for example) and programs might not recognize the temporarily acquired capabilities, though we believe that most (including, most importantly, the file system's file security checking) will.
- Normally, when you run a program or execute a command file, MPEX (and MPE) first try to find a temporary file with that name and only look for a permanent file if the temporary file doesn't exist. This would make it possible for a user to violate security by creating a temporary file with the same name as a command file or program that was to be executed while a WITHCAPS was in effect. To avoid this, MPEX, LOGON and STREAMX will, while a WITHCAPS is in effect, only execute permanent command files and programs (unless you explicitly indicate otherwise in the command file/menu/job stream by using a file equation with a ,OLDTEMP on it).
- By the same token, we do not execute UDCs when WITHCAPS is in effect. The reason for this is that the operating system resolves UDCs by scanning the USER level UDCs first, then the ACCOUNT level and finally the SYSTEM level. If we were to execute UDCs while a user had additional capabilities, all a user would have to do to circumvent security would be to set a USER level UDC with the same name as an ACCOUNT or SYSTEM level UDC or even an MPE command — therefore we don't allow this.
- Also for security reasons, MPEX will not honor MPEXCRITRESTRICTxxx variables (which restrict MPEX fileset handling commands to operate only on certain types of files) and VESOFTDEFAULTxxx variables (which set defaults for %ALTFILE, %COPY, %PRINT, :RUN and implied RUN parameters) while a WITHCAPS is in effect.
- You cannot use WITHCAPS to acquire PM, PH, MR, DS, IA and BA capabilities; fortunately, you probably wouldn't want to, since the most useful capabilities to acquire are SM, AM and OP, and possibly also AL and GL.
- As we said before, this feature is only available in MPEX if you are user of SECURITY.

%XEQ

Syntax:

```
%XEQ commandfilename [parameterlist]
%XEQ progfilename [[INFO=] infostring] [; [PARM=] parmvalue]
```

Examples:

```
%XEQ MYPROG INFO="MYFILE";PARM=%10001
%XEQ MYPROG "MYFILE",%10001
```

In principle, "%XEQ *xxx yyy*" does the same thing as "*xxx yyy*" — it finds a program file or a command file named "*xxx*" and executes it, passing to it the parameters "*yyy*". (See ["Implied run"](#) and ["Command files"](#) in the MPEX Commands chapter for more details.)

Why have the %XEQ command at all? The only reason for it is if "*xxx*" happens to be the name of an MPE or MPEX commands or a UDC.

Remember that the sequence in which MPEX tries to execute commands is:

- UDCs,
- MPEX commands,
- built-in MPE commands,
- and finally command files or implied :RUNs.

If the command file or program you want to execute has the same name as a UDC or an MPE or MPEX command (e.g. "INPUT" and "FCOPY" in the examples above), you need to use the %XEQ command.

Note that %XEQ honors the HPPATH variable just like normal command file execution and implied :RUNs do.

Appendix A: POSIX Compatibility issues

The following is a list of everything that we believe could be considered an "incompatibility" with pre-POSIX versions of MPEX. We don't believe any of these changes should have negative impact on you, but we want to make you aware of them.

For more details on any of these features, please see the main entry for that feature in the MPEX User Manual.

Important note for users of MPEX/iX 4.5:

Version 4.5 of the MPE/iX operating system was the first version to include POSIX-compliant extensions, and was sent primarily to software developers and users specifically interesting in Beta testing these extensions. MPE/iX version 5.0 is the first generally distributed version to include POSIX extensions. Because of a number of limitations in MPE/iX 4.5, with the release of 5.0 we no longer support version 4.5 of the operating system.

Visual VEMODIFY (for %REDO) extends over 1 screen line

The VEMODIFY function, used for %REDO and HOOKed programs, when editing in visual mode, has been enhanced to allow lines containing up to 150 characters to be edited. Characters inserted on the first line will insert-wrap to the second line when they reach the last screen column, and characters deleted from the first line will pull back characters from the second line.

By default, VEMODIFY assumes that your terminal is 80 columns wide, and uses cursor-positioning escape sequences that won't work correctly if you have a 132 column wide terminal.

If you switch between 80 and 132 character-wide modes, you can now configure VEMODIFY to use width-independent escape sequences to position the cursor. To do this you must setup the JCW VESOFTVEMODIFYEXPANDOK=1 before using VEMODIFY the first time, or issue the %INITREDO command after setting the JCW.

VEMODIFY uses much longer escape sequences when you configure it this way, so we suggest you only use this mode when connected over a high speed line.

Type "%HELP VEMODIFY" for more details on this topic.

Using the continue ("%") character

MPEX handles continuation lines the same way MPE does on a CLASSIC system, i.e. there will be one space inserted where the continuation character appears in the line. MPE/iX however, does not insert a space, and also strips any leading spaces from the next line. In order to make MPEX behave the same way, you must set the JCW VESOFTCONTINUENOSPACE=1. Be aware that this may cause files that worked before to fail.

For example, the following IF command will fail:

```
IF a=b AND&
    j=k THEN
    ECHO Test true
ENDIF
```

as the letter "j" will be appended to the "AND" operator. Both MPEX and MPE/iX will report an error when trying to evaluate the variable "ANDj".

Type "%HELP CONTINUE" for more details on this topic.

"@.@.@." vs. "/" on systems that support POSIX

Historically, "@.@.@." meant every file on the system. With the release of MPE/iX 4.5, however, it became necessary to use "/" to specify all files on the system. ("@.@.@." was limited to all MPE files on the system — files stored in subdirectories under the root or other MPE groups would not qualify as part of "@.@.@.")

With the release of MPE/iX 5.0, MPE's :STORE command was modified to treat "@.@.@." and "@.@.MYACCT" as equivalent to "/" and "/MYACCT/" respectively. This is currently the only HP supplied utility that acts in this manner – the :LISTF command only list MPE files, and the :LISTFILE command only lists POSIX files if you specify a POSIX-syntax fileset (one beginning with a "." or "/"). Other HP commands and utilities act in a similar manner, i.e. only finding files in the MPE or POSIX namespace depending upon the fileset given in the command.

MPEX commands that accept filesets, however, will accept either a POSIX style fileset, an MPE style fileset, or a mixture of MPE and POSIX filesets and automatically format the output as appropriate. In addition, %LISTF @.GROUP will display any POSIX filenames stored in that group (but not in any subdirectories of that group).

Type "%HELP HFS" or "%HELP POSIX" for more details on this topic.

LISTF format changes

A number of extensions have been made to the output of MPEX's %LISTF modes (on both POSIX and pre-POSIX systems), as summarized here:

All %LISTF modes:

On POSIX system, unlike the MPE/iX :LISTF command, the MPEX %LISTF command shows all POSIX files, including lower and mixed-case filenames, filenames containing special characters, and long filenames.

If you wish to inhibit the display of all POSIX files (and emulate the behavior of MPE's :LISTF command), issue the following command:

```
%SETVAR VESOFTNOTALWAYSPOSIX TRUE
```

%LISTF,-1:

On POSIX systems, the output of %LISTF,-1 has been extended to display 332 words of the filelabel, as opposed to the 256 words displayed by the MPE/iX :LISTF,-1.:

If you wish to emulate the behavior of MPE's :LISTF command, issue the following command:

```
%SETVAR MPEXLISTFLABELSIZE 256
```

%LISTF,2:

The output of %LISTF,2 has been extended to display the number of days since a file was last accessed and modified. The new fields are displayed to the right of the traditional :LISTF,2 fields.

If you wish to inhibit the display of the two new fields, issue the following command:

```
%SETVAR MPEXLISTFNODAYS TRUE
```

%LISTF,-3 and %LISTF,3:

The output of %LISTF...,3 and %LISTF..., -3 has been modified on POSIX systems to include information on a file's OWNER and GROUPID. In addition MPEX's %LISTF also reports the numeric UID and GID equivalents (when the filename is specified using HFS-syntax), as well as the POSIX State-Change timestamp, and the target filename of a link file.

The "**SECURITY IS ON" field now also indicates if a file has an ACD assigned, and shows (on POSIX systems) if the ACD is required.

On Spectrum systems, volume restrictions are now accurately displayed, as well as an indication if the file is a system bootfile, located on device \$SYSTEM_MASTER.

File attributes: CREATOR, CREATORACCOUNT, and OWNER

There are two new attributes related to the CREATOR attribute, these are CREATORACCOUNT and OWNER. The CREATOR is the MPE USER ID of the owner of the file, (just as it has always been), while the CREATORACCOUNT is the MPE ACCOUNT name of the owner of the file and the OWNER is both the CREATOR and CREATORACCOUNT attributes in the form "user.account".

When CREATOR is used within extended fileset selection, special logic is used to allow it to match either the traditional "user" or the new "user.account" posix styles of representing the file creator. When OWNER is used within extended fileset selection, it will only match strings using the "user.account" format.

In other words, (CREATOR="MANAGER.SYS") works just as well as (OWNER="MANAGER.SYS"), (CREATOR="CLERK") works the same as it always has, while (OWNER="CLERK") will never match anything.

When these attributes are recovered using the VEFINFO(...) function, CREATOR just returns the USER info while OWNER returns both the user and account. This is very useful when trying to determine if the creator of a file still exists.

In the %ALTFILE section, we gave this example:

```
%ALTFILE @.@(NOT USEREXISTS(CREATOR+"."+ACCOUNT));CREATOR=MANAGER
```

Using the OWNER attribute instead, this can be simplified as follows:

```
%ALTFILE @.@(NOT USEREXISTS(OWNER));CREATOR=MANAGER.ACCT
```

Type "%HELP CREATOR" for more details on this topic.

Extended filesets using CREATOR= and CODE=

In order to be compatible with older versions of our software, version 2.4 of our software automatically quoted the value specified for CREATOR and CODE when used in an extended selection. A warning was issued, however, that we were doing this and that at some point in the future this might change. This is inconsistent with the rest of our software as these attributes are defined as STRING attributes, but could not be compared to STRING variables. (Since we were automatically quoting the comparison value, using the name of a string variable would not work as we would try to quote the name of the variable instead of comparing the CREATOR or CODE to the VALUE of the variable.)

Starting in version 27., when we see an extended selection of (CREATOR=MYVAR), we will first check to see if there is a (string) variable called MYVAR and if so, use the value of the variable for the comparison. If MYVAR doesn't exist, then we will issue the old warning and search for files created by the user "MYVAR". For the CODE attribute, however, we treat valid values as reserved words and issue the warning that you should use a quoted value. For instance:

```
%SETVAR A "BASFP"  
%LISTF @.SOURCE(CODE=A)
```

will work, but

```
%SETVAR BASFP "BASFP"  
%LISTF @.SOURCE(CODE=BASFP)
```

will generate a warning message.

Errors and warnings in STDLIST(...) honor I/O redirection

:In prior versions, errors or warnings detected during execution of the STDLIST(...) function would ALWAYS display on your terminal, even if the "command" in the STDLIST(...) function had I/O redirection specified. Starting in version 27., errors or warnings will be directed to any I/O redirection file as specified in the STDLIST(...) function call. For example:

```
%SETVAR XFILE STDLIST("SHOWVAR BADVARIABLENAME >MYFILE")
```

would display an error message in 2.4, even though the "result" of the SHOWVAR command was being redirected.

Starting in version 27., the file MYFILE will actually contain the text of the error message. This will only effect you if you expect the file to be empty if there is an error. In both versions,

however, the variable `HPCIERR` is set to indicate whether or not an error occurred in the command passed to the `STDLIST(...)` function.

Type `%HELP STDLIST` for more details on this topic.

ISxxxCAP variables/functions fixed

In previous versions, some `ISxxxCAP` variables would return a 1 or 0 instead of `TRUE` or `FALSE` as documented. These still worked in an extended selection as we treated a 1 as `TRUE` and 0 as `FALSE`, but direct comparisons would fail. If you worked-around this problem in a job stream or command file, you will have to fix your files to compare against `TRUE` or `FALSE` as appropriate.

Type `%HELP ATTRIBUTE` for more details on this topic.

%ABORTJOB sched can abort STREAMX scheduled jobs

The scope of the `SCHED` job attribute has been extended to encompass both MPE scheduled jobs and `STREAMX` scheduled jobs (`STREAMX SCHEDULING` features — repetitive and conditional job submission — are described in the `SECURITY MANUAL`.)

As a result of this change, issuing the MPEX command `%ABORTJOB SCHED` will now result in the loss of all `STREAMX` scheduled jobs (including the specification of when the job is to be launched).

If you wish to limit the scope of `SCHED` to MPE scheduled jobs only, you should use the MPEX command `%ABORTJOB @J&SCHED` instead.

Type `%HELP ABORTJOB` or `%SEC HELP STREAMX` for more details on this topic.

%ALARM starts the BACKG job if it's not already running

The `%ALARM` command will start the `%BACKG` background jobstream if the job is not already running. Although this is the normal behavior for `SM` or `OP` users, the `%ALARM` command has been enhanced to start the `BACKG` job (if necessary) when used by `ANY` user. However, if the `%ALARM` command starts the background job, then only the `%ALARM` task is started — any other "pending" tasks remain pending until a normal `%BACKG STARTJOB` command is issued.

Type `%HELP ALARM` or `%HELP BACKG` for more details on this topic.

%ALTFILE program;CAP=-xx,yy,zz changed

When using `;CAP=` on an `%ALTFILE` or `%COPY` command, "-" signs do not need to be repeated. i.e., `;CAP=PH,-MR,DS` is equivalent to `;CAP=+PH,-MR,-DS`. Previous versions of MPEX treated all unsigned capabilities as having an implied "+", i.e. it would have ADDED PH and DS while REMOVING MR. Similarly, specifying `;CAP=-BA,PM,PH,+MR,DS` would be treated as `;CAP=-BA,-PM,-PH,+MR,+DS`. In other words, unsigned capabilities retain the same sign as the previous capability listed.

Type `%HELP ALTFILE` for more details on this topic.

%ALTFILE ;CREATOR changes on POSIX systems

MPEX has traditionally allowed a file's CREATOR to be changed to any value, regardless of whether the user name specified actually exists on your system. On POSIX systems MPE/iX has its own :ALTFILE command, which allows a file's OWNER (and GROUPID) to be changed to any User (and Account) that are currently configured on your system.

To maintain compatibility with MPE/iX, when used on a POSIX system, MPEX's %ALTFILE command will now check that supplied CREATOR names actually exist, and will fail if they don't. If you need to use %ALTFILE to set a file's CREATOR or GROUPID to values that don't currently exist on a POSIX system, use the new ;NOCHECK keyword.

For example, if the following ALTFILE command fails as shown:

```
%ALTFILE file; CREATOR=MISSING.SYS
Specified owner does not exist in the user database. (CIERR 820)
```

Then you can force this to work as follows:

```
ALTFILE file; CREATOR=MISSING.SYS ;NOCHECK
```

%SET CREATORPROTECT now enforced on POSIX systems

MPEX's default behavior has always allowed users of any capability level to use the %ALTFILE command to change the CREATOR of any file that they have read and write access to. The %SET CREATORPROTECT command allowed system managers unhappy with this default to prevent ordinary users from changing CREATOR names on any file, and to prevent account managers from altering files outside their account. With MPE/iX 5.0, MPE/iX has its own :ALTFILE command, which only allows users with appropriate privileges to change the fully-qualified OWNER of a file.

To maintain compatibility with MPE/iX, when used on a POSIX system, MPEX now always enforces %SET CREATORPROTECT. The error messages displayed if a user fails in an attempt to change a file's CREATOR name have also changed to be compatible with MPE/iX.

Type "%HELP CREATORPROTECT" for more details on this topic.

VEFxxx functions, changes for SM users

SM users can now access PRIV files using the VEFxxx functions. When the correct file code is specified as part of the file parameter of the VEFOPEN, VEFREAD, and VEFWRITE functions, an SM user can open, read, and write PRIV files.

Warning: PRIV files are privileged for a reason! Writing to PRIV files (IMAGE Database files, for example), may corrupt the internal structure of the file and make them useless to your application. Use this feature with care — VESOFT, Inc. cannot repair files corrupted (whether PRIVileged or not) by misuse of these functions.

MPE error messages down-shifted

To remain compatible with MPE/iX 5.0, we have down-shifted the text associated with many of the MPE error messages that MPEX displays. Users on pre-MPE/iX 5.0 releases will notice that many (but not all) of the messages we display are now down-shifted.

Expressions: TYPEOF() enhanced for special types

The `typeof()` function has been enhanced to recognize MPEX types of `REAL`, `DATE`, and `TIME`. These types return 101, 102, and 103 respectively.

Note that these values are returned only if the argument to the function is an `EXPRESSION` and not a simple variable. Also, setting the result of an expression to a variable will still convert the result into a `STRING` for these types of expressions, so:

```
%CALC TYPEOF(TODAY+10)
```

would produce a different result than:

```
%SETVAR TESTDATE TODAY+10
%CALC TYPEOF (TESTDATE)
```

QEDIT "jumbo" files supported by %PRINT & %LISTF,ID

Version 4.3 of Robelle's QEDIT editor added support for extended size QEDIT files, known as "jumbo" files. These files may contain up to 1000 characters per line, and be over 65,535 lines long. Previous versions of MPEX were not able to correctly display the contents of these files (which have the same "111" filecode as standard QEDIT files).

This version of MPEX fully supports these "jumbo" files: the `%PRINT` command understands their internal format, and `%LISTF, ID` correctly displays their logical line length, file size and file type. We've also added two new file attributes: `ISQEDIT` qualifies all QEDIT files, `ISJUMBO` qualifies "jumbo" QEDIT files only.

HPREDOSIZE variable changed to MPEXREDOSIZE

There are several options you can set for the REDO facility (which apply equally to the REDO facility available from MPEX HOOKed programs). They are set by defining various specially-named JCWs and/or variables. These JCWs must be set before issuing a `%INITREDO` command or running a HOOKed program within which you wish them to take effect.

In prior versions of our software, the variable that you would use to set the size of the history file was `HPREDOSIZE` — the same variable MPE/iX uses. However, due to the way MPE/iX handles the `HPREDOSIZE` variable, setting `HPREDOSIZE` outside of MPEX has no effect, so we now use the variable `MPEXREDOSIZE`. Setting `HPREDOSIZE` inside MPEX is translated to `MPEXREDOSIZE`, so users who set `HPREDOSIZE` in their `MPEXMGR` files prior to issuing the `%INITREDO` command will not experience any problems.

Appendix B: VESoft EXPRESSIONS

Introduction

In a number of places, we talk about "expressions". These expressions are very powerful things; they let you do various tests, calculations, date arithmetic and so on. They support a number of different data types, operators and functions and may be used to process strings, integers, dates, times, boolean conditions and even real numbers.

In the simplest case, an expression is very much like an expression in BASIC, COBOL, PASCAL, etc. For instance,

```
1+2*3+4
```

comes out exactly the way you expect it (11); the normal precedence of the operators is respected (* first, then +).

Similarly,

```
LEN("FOOBAR")
```

calculates the length of the string "FOOBAR" (which is 6);

```
UPS("Testing")
```

returns "TESTING", the upshifted value of the string "Testing";

```
MIN(123,456)
```

returns 123, the minimum of MIN's parameters (123 and 456).

Variables

In addition to the functions (MIN, UPS, LEN, etc.), the operators (+, -, *, /, etc.) and the constants (123, "Testing", etc.), you can also use variables. Variables are what makes expressions so very powerful because they're **adapted to the program in which they're used**:

- In MPEX selection criteria you can refer to variables that represent various attributes of the current file (e.g. CODE, SECTORS, ACCDATE, etc.); for instance, an MPEX selection criterion may say

```
ISPROG and ACCDATE<1/1/1997
```

- In MPEX's %CALC, %SETVAR and %IF you can refer to any JCW, any variable previously set by a %SETVAR, and any one of a number of predefined variables that let you figure out things like the logon user ID, the logon account, the current day of week, the current job or session limit, etc.
- In SECURITY's LISTLOG and LISTUSERS options, you can refer to various fields of the SECURITY log record and SECURITY user record, respectively. For instance, you might try to find all log or user records that match the criterion

```
USER MATCHES "X@" and ACCOUNT<>"SYS"
```

- In STREAMX, you can refer to any JCW, any predefined variable (see below), AND any variable set by a STREAMX ::PROMPT or ::ASSIGN command. Thus, a STREAMX's CHECK=... expression might say

```
RPTCODE='XX' or RPTCODE='Y7' or RPTCODE[0:1]='Z'
```

where RPTCODE is the name of the variable being ::PROMPTed for.

The differences between the kinds of variables acceptable to the different programs are intentional, since in each case expressions have different purposes — sometimes to operate on file attributes, sometimes to select log records, sometimes to work on STREAMX variables.

Data types

In the examples we showed above, we saw three data types:

- **INTEGERS.** These are simple numbers from -2147483648 to +2147483647 (i.e. 32-bit or "DOUBLE" integers). You can specify them as decimal, as octal (prefixed by a %, e.g. %12345670) or as hexadecimal (prefixed by a \$, e.g. \$89ABCDEF).
- **STRINGS.** These are sequences of characters — string constants can be enclosed in either double quotes ("ABC") or single quotes ('ABC') although the quote types must obviously match (e.g. "ABC" and 'ABC' are both invalid). You may embed quotes into a quoted string by specifying the quote character twice, e.g. "ABC""DEF" stands for the string ABC"DEF and 'ABC' 'DEF' stands for the string ABC'DEF; however, "ABC" 'DEF' stands for the string ABC' 'DEF. The empty string (length 0) is "" (or ' ').
- **DATES.** These are dates from 1 January 1901 to 31 December 1999 with a 2-digit year or from 1 January 1600 onward (into the year 2000 and beyond) with a 4-digit year. Anywhere that you can input a 2-digit year, you can also input a 4-digit year!

Any time you are prompted for a date (e.g. by ::PROMPT in STREAMX jobs, PROMPT in VEMENUs or the READDATE function) and in the DATEPARSE and VALIDDATE functions, you can enter it either with or without slashes ("/"); however, when you are specifying a date constant in an expression (rather than being prompted for it), you must include the slashes (to distinguish it from an integer).

Date constants are usually (in STREAMX and SECURITY) specified in [YY]YY/MM/DD format (e.g. 90/12/21 or 2024/05/14), although they can be output in any format you please. By default, date output and formatting are in 2-digit year format, with slashes.

In MPEX (for historical reasons), dates are by default input in MM/DD/[YY]YY format (12/31/99, 11/05/2022 or even 102283), but you can use the %SET DATE command to change the default input format to [YY]YY/MM/DD or DD/MM/[YY]YY (the European way)

Several other data types are also supported:

- **BOOLEAN VALUES** (also known as LOGICALS). A boolean value may be either TRUE or FALSE — various operators like =, <, >, <>, >=, <=, AND, OR, NOT and so on return boolean values. In some places, e.g. an MPEX selection condition, a SECURITY LISTLOG or

LISTUSERS selection condition or a STREAMX ::PROMPT...;CHECK=... expression a boolean value is explicitly required; if you say in MPEX

```
%!LISTF @.@.@ (2+3)
```

then MPEX will print you an error message since the selection condition should be a boolean expression, not an integer like 2+3.

- **REAL NUMBERS.** Real numbers (32-bit single-precision, "HPREAL") are also supported — saying

```
%CALC 1.2 * 3.4
```

in MPEX will give you the right value (4.08). The range of the magnitude of non-zero real values is $9.9999 \times (10^{-75})$ through $1.000004 \times (10^{76})$; real numbers are accurate to 6.9 decimal places. Real constants can be entered in fixed-point format (1.234) or in the exponential format known and loved by all you FORTRAN programmers (5.6E10).

- **TIMES.** You may also use the time-of-day in expressions. 12:00AM or 00:00T is midnight; 12:00PM or 12:00T is noon.

A time constant is specified by saying one of

hour:minute AM	e.g. 10:25 AM	
hour:minute PM	e.g. 6:45 PM	
hour:minute T	e.g. 17:07 T	(24-hour clock)

and times may be compared (RSTTIME>12:15AM) or printed.

Types are very important because the same operators (e.g. + and -) can have very different effects when operating on, say, integers, strings and dates:

10 + 20	is 30 (simple addition)
"FOO" + "BAR"	is "FOOBAR" (string concatenation)
91/2/20 + 10	is 91/3/2 (date increment)
1.2 + 3.4	is 4.6 (also addition, but on real numbers)
TRUE + FALSE	is an error (you can't add booleans);
"FOO" + 10	is an error (you can't add numbers to strings);

Always keep in mind what types you're operating on, since that will influence the meaning of the operation. Also note that although all of VESOFT's programs (MPEX, SECURITY and VEAUDIT) allow expressions of all types, not all types may be used to define variables — MPEX variables can only be integers, strings or booleans and STREAMX variables can only be integers, strings or dates.

If you try to set an MPEX variable to a real, date or time value, (like 1.2, 90/11/05 or 11:59AM), the result will be converted to a string ("1.2", "90/11/05" and "11:59AM") before assigning it to the variable — this is for compatibility with MPE/iX.

Appendix B

In all of the examples in this appendix (unless otherwise noted):

I, I1, I2, I3	Integer variables, constants and/or expressions
B, B1, B2	Boolean variables, constants and/or expressions
R, R1, R2	Real variables, constants and/or expressions
S, S1, S2, S3	String variables, constants and/or expressions
D, D1, D2	Date variables, constants and/or expressions
T, T1, T2, T3	Time variables, constants and/or expressions

The type of the result returned by the various functions are indicated by (in order of their appearance):

[INT]	Integer
[BOOL]	Boolean
[STR]	String
[DATE]	Date
[REAL]	Real
[TIME]	Time
[NONE]	No function return
[???	Function return type depends on attribute type requested

Integer and boolean operators and functions

The following operators and functions work on integers and boolean values:

Operator or Function	Type	Operation Performed
$I1 + I2$	[INT]	Addition.
$I1 - I2$	[INT]	Subtraction.
$I1 * I2$	[INT]	Multiplication.
$I1 / I2$	[INT]	Integer division, truncates result. 20/3 is 6.
$I1 \text{ MOD } I2$	[INT]	Integer remainder. 20 MOD 3 is 2.
$-I$	[INT]	Negation. -10 is, of course, -10.
ABS (I)	[INT]	Absolute value of I . ABS(-10) is 10. ABS(10) is 10.

Operator or Function	Type	Operation Performed
MIN (<i>I1</i> [, <i>I2</i> ...])	[INT]	Returns the smallest of one (or more) integers.
MAX (<i>I1</i> [, <i>I2</i> ...])	[INT]	Returns the largest of one (or more) integers.
<i>I1</i> ^ <i>I2</i>	[INT]	Returns <i>I1</i> raised to the power <i>I2</i> . 10^3 is 1000, 2^10 is 1024.
RANDOM (<i>I</i>)	[INT]	Returns a random number in the range 0 to <i>I</i> -1.
<i>I1</i> = <i>I2</i>	[BOOL]	Tests whether <i>I1</i> is equal to <i>I2</i> .
<i>I1</i> < <i>I2</i>	[BOOL]	Tests whether <i>I1</i> is less than <i>I2</i> .
<i>I1</i> > <i>I2</i>	[BOOL]	Tests whether <i>I1</i> is greater than <i>I2</i> .
<i>I1</i> <> <i>I2</i>	[BOOL]	Tests whether <i>I1</i> is not equal to <i>I2</i> .
<i>I1</i> >= <i>I2</i>	[BOOL]	Tests whether <i>I1</i> is greater than or equal to <i>I2</i> .
<i>I1</i> <= <i>I2</i>	[BOOL]	Tests whether <i>I1</i> is less than or equal to <i>I2</i> .
BETWEEN (<i>I1</i> , <i>I2</i> , <i>I3</i>)	[BOOL]	Tests whether <i>I1</i> is between <i>I2</i> and <i>I3</i> , i.e. <i>I1</i> >= <i>I2</i> AND <i>I1</i> <= <i>I3</i> ; BETWEEN(<i>I</i> , 1, 10) is TRUE if <i>I</i> is between 1 and 10 (inclusively).
ODD (<i>I</i>)	[BOOL]	Tests whether <i>I</i> is odd rather than even. ODD(1) is TRUE, ODD(2) is FALSE.
<i>B1</i> AND <i>B2</i>	[BOOL]	TRUE if <i>B1</i> and <i>B2</i> are both TRUE.
<i>B1</i> OR <i>B2</i>	[BOOL]	TRUE if either <i>B1</i> or <i>B2</i> (or both) is TRUE.
<i>B1</i> XOR <i>B2</i>	[BOOL]	TRUE if either <i>B1</i> or <i>B2</i> (but not both) is TRUE.
NOT <i>B</i>	[BOOL]	TRUE if <i>B</i> is FALSE, FALSE if <i>B</i> is TRUE.
<i>B1</i> & <i>B2</i>	[BOOL]	Synonym for <i>B1</i> AND <i>B2</i> .
<i>B1</i> <i>B2</i>	[BOOL]	Synonym for <i>B1</i> OR <i>B2</i> .
TRUE, FALSE	[BOOL]	Boolean constants.
ON, OFF	[BOOL]	Boolean constants; ON = TRUE, OFF = FALSE.
<i>I1</i> [<i>I2</i> : <i>I3</i>]	[INT]	Extracts <i>I3</i> bits (starting with bit # <i>I2</i>) from the integer value <i>I1</i> ; for instance %1234[10:3] is 3. Note that the highest-order bit (since <i>I1</i> is a 32-bit integer) is bit -16 (!) and the lowest-order bit is bit 15. This way, when you extract bits from single integers,

Operator or Function	Type	Operation Performed
		you can use the 0-to-15 bit numbering; bits -16 to -1 refer to the high-order word.
DBBUILD (<i>I1</i> , <i>I2</i>)	[INT]	Constructs a double integer with high-order word <i>I1</i> and low-order word <i>I2</i> . DBUILD(1,2) is 65538.
HIGHORDER (<i>I</i>)	[INT]	Returns the high-order word of <i>I</i> .
LOWORDER (<i>I</i>)	[INT]	Returns the low-order word of <i>I</i> .
BNOT <i>I</i>	[INT]	Returns the Bit-wise NOT of <i>I</i> .
<i>I1</i> BAND <i>I2</i>	[INT]	Returns a Bit-wise AND of the integers <i>I1</i> and <i>I2</i> .
<i>I1</i> BOR <i>I2</i>	[INT]	Returns a Bit-wise OR of the integers <i>I1</i> and <i>I2</i> .
<i>I1</i> BXOR <i>I2</i>	[INT]	Returns a Bit-wise XOR of the integers <i>I1</i> and <i>I2</i> .
<i>I1</i> LSL <i>I2</i>	[INT]	Logical Shift Left the integer <i>I1</i> by <i>I2</i> bits.
<i>I1</i> LSR <i>I2</i>	[INT]	Logical Shift Right the integer <i>I1</i> by <i>I2</i> bits.
<i>I1</i> CSL <i>I2</i>	[INT]	Circular Shift Left the integer <i>I1</i> by <i>I2</i> bits.
<i>I1</i> CSR <i>I2</i>	[INT]	Circular Shift Right the integer <i>I1</i> by <i>I2</i> bits.
<i>I1</i> ASL <i>I2</i>	[INT]	Arithmetic Shift Left the integer <i>I1</i> by <i>I2</i> bits.
<i>I1</i> ASR <i>I2</i>	[INT]	Arithmetic Shift Right the integer <i>I1</i> by <i>I2</i> bits.
INTEGER (<i>S</i>)	[INT]	Returns the value of the 16-bit integer corresponding to the 2 bytes in <i>S</i> (if <i>S</i> is a 2-character string) or the 32-bit integer corresponding to the 4 bytes in <i>S</i> (if <i>S</i> is a 4-character string). Triggers an error if <i>S</i> is neither 2 nor 4 characters long. Useful for extracting integer values from records read from files using the VEFREADxxx functions.
INTEGERPARSE (<i>S</i>)	[INT]	Converts the string <i>S</i> to an integer; if it's not a valid integer, aborts with an error message.
VALIDINTEGER (<i>S</i>)	[BOOL]	Returns TRUE if <i>S</i> is a valid integer, FALSE if it isn't.

String operators and functions

The following operators and functions work on strings:

Attribute	Type	Operation Performed
$S1 + S2$	[STR]	Concatenates two (or more!) strings. "AB" + "CD" is "ABCD".
$S1 - S2$	[STR]	Removes first occurrence of string $S2$ from string $S1$. "TOTO" - "O" is "TTO".
$I * S$	[STR]	Repeats I times the string S . $3 * \text{"AB"}$ is "ABABAB".
$S [I1:I2]$	[STR]	Extracts $I2$ characters of string S starting with character # $I1$ (counting the first character as #0); "FOOBAR" [2:3] is "OBA".
STRRTRIM ($S1$ [, $S2$])	[STR]	Returns $S1$ with all trailing occurrences of the character specified by $S2$ (which defaults to " ") removed. STRRTRIM ("FOO ") is "FOO"; STRRTRIM ("ABC---", "-") is "ABC".
STRLTRIM ($S1$ [, $S2$])	[STR]	Returns $S1$ with all leading occurrences of the character specified by $S2$ (which defaults to " ") removed. STRLTRIM (" FOO") is "FOO"; STRLTRIM ("---ABC", "-") is "ABC".
RTRIM($S1$ [, $S2$])	[STR]	Same as STRRTRIM.
LTRIM($S1$ [, $S2$])	[STR]	Same as STRLTRIM.
STRLDROP (S , I)	[STR]	Returns the string S with the first I characters removed. STRLDROP("INFLAMMABLE", 2) is "FLAMMABLE".
STRRDROP (S , I)	[STR]	Returns the string S with the last I characters removed. STRRDROP("ABSOLUTELY NOT", 4) is "ABSOLUTELY".
LEN (S)	[INT]	Returns the length of string S . LEN("NOSE") is 4.
POS ($S1$, $S2$ [, I])	[INT]	Returns the position of the I th (starting with 1) occurrence of $S1$ in $S2$. If I is negative, returns the position of the ($-I$)th occurrence of $S1$ in $S2$, counting from the right. If $S1$ is not found in $S2$, returns 0. (I defaults to 1).

Attribute	Type	Operation Performed
		POS("A", "SPARK") is 3; POS("IS", "MISSISSIPPI", 2) is 5; POS("PE", "PEPPERONI", -2) is 1.
UPS (<i>S</i>)	[STR]	Returns <i>S</i> with all alphabetic characters upshifted. UPS("Test") is "TEST".
DWNS (<i>S</i>)	[STR]	Returns <i>S</i> with all alphabetic characters downshifted. DWNS("Test") is "test".
STRCHANGE (<i>S1</i> , <i>S2</i> , <i>S3</i>)	[STR]	Returns the string <i>S1</i> in which all occurrences of the string <i>S2</i> have been changed to the string <i>S3</i> . STRCHANGE("TESTING", "T", "XX") is "XXESXXING".
STRCOUNT (<i>S1</i> , <i>S2</i>)	[INT]	Returns the number of occurrences of <i>S1</i> in <i>S2</i> . STRCOUNT("P", "Pepper") is 1; STRCOUNT("SS", "MISSISSIPPI") is 2.
RPT (<i>S</i> , <i>I</i>)	[STR]	Returns string <i>S</i> repeated <i>I</i> times. If <i>I</i> <0, reverses string <i>S</i> before repeating it. RPT("ABC", 3) is "ABCABCABC"; RPT("ABC", -3) is "CBACBACBA".
CHR (<i>I</i>)	[STR]	Returns a single-character string containing ASCII character <i>I</i> . CHR(33) is "!"; CHR(27) is the [Escape] character.
ORD (<i>S</i>)	[INT]	Returns integer ASCII value of the first character of <i>S</i> . ORD("!") is 33.
HEX (<i>I</i>)	[STR]	Returns value of <i>I</i> converted to a hexadecimal string. HEX(32767) is "\$7FFF".
OCTAL (<i>I</i>)	[STR]	Returns value of <i>I</i> converted to an octal string. OCTAL(32767) is "%77777".
STR (<i>S</i> , <i>I1</i> , <i>I2</i>)	[STR]	Returns <i>I2</i> characters of string <i>S</i> starting with character # <i>I1</i> (counting the first as #1). STR("FOOBAR", 2, 3) is "OOB".
LFT (<i>S</i> , <i>I</i>)	[STR]	Returns the <i>I</i> leftmost characters of string <i>S</i> . LFT("FOOBAR", 2) is "FO".
RHT (<i>S</i> , <i>I</i>)	[STR]	Returns the <i>I</i> rightmost characters of string <i>S</i> . RHT("FOOBAR", 2) is "AR".
ALPHA (<i>S</i>)	[BOOL]	TRUE if <i>S</i> consists entirely of alphabetic characters.
NUMERIC (<i>S</i>)	[BOOL]	TRUE if <i>S</i> consists entirely of numeric characters.

Attribute	Type	Operation Performed
ALPHANUM (<i>S</i>)	[BOOL]	TRUE if <i>S</i> consists entirely of alphabetic and/or numeric characters.
MAXCONSECUTIVE (<i>S</i>)	[INT]	Returns the maximum number of consecutive occurrences of any characters in <i>S</i> . MAXCONSECUTIVE("ABBCCCDDE") is 3, since the character 'C' occurs 3 times in a row.
<i>S1</i> = <i>S2</i>	[BOOL]	Tests if <i>S1</i> is equal to <i>S2</i> .
<i>S1</i> < <i>S2</i>	[BOOL]	Tests if <i>S1</i> is less than <i>S2</i> .
<i>S1</i> > <i>S2</i>	[BOOL]	Tests if <i>S1</i> is greater than <i>S2</i> .
<i>S1</i> <> <i>S2</i>	[BOOL]	Tests if <i>S1</i> is not equal to <i>S2</i> .
<i>S1</i> >= <i>S2</i>	[BOOL]	Tests if <i>S1</i> is greater than or equal to <i>S2</i> .
<i>S1</i> <= <i>S2</i>	[BOOL]	Tests if <i>S1</i> is less than or equal to <i>S2</i> .
<i>S1</i> MATCHES <i>S2</i>	[BOOL]	Tests if <i>S1</i> matches the MPE-like pattern <i>S2</i> , where "?" character in <i>S2</i> matches a single alphanumeric character, "#" matches a single numeric character, "@" matches any number (0 or more) alphanumeric characters; "[...]" range operator (see "File ranges" in the MPEX User Manual).
<i>S1</i> NMATCHES <i>S2</i>	[BOOL]	Same as NOT (<i>S1</i> MATCHES <i>S2</i>).
MIN (<i>S1</i> [, <i>S2</i> ...])	[STR]	Returns the "minimum" of one (or more) strings. "Minimum" means alphabetically, not according to length, thus MIN("AAA", "ZZ") is "AAA"; MIN("BIG", "SMALL", "MEDIUM") is "BIG".
MAX (<i>S1</i> [, <i>S2</i> ...])	[STR]	Returns the "maximum" of one (or more) strings. "Maximum" means alphabetically, not according to length, thus MAX("AAA", "ZZ") is "ZZ"; MAX("BIG", "SMALL", "MEDIUM") is "SMALL".
BETWEEN (<i>S1</i> , <i>S2</i> , <i>S3</i>)	[BOOL]	Tests whether <i>S1</i> is between <i>S2</i> and <i>S3</i> , i.e. <i>S1</i> >= <i>S2</i> and <i>S1</i> <= <i>S3</i> ; for instance, BETWEEN(<i>S</i> , "JA", "LM") is TRUE if <i>S</i> is between "JA" and "LM" (inclusively).
TOKEN (<i>S1</i> , <i>S2</i>)	[STR]	Returns the first "token" of <i>S1</i> , where <i>S2</i> is a string containing the token delimiters; a "token" is the set of characters, excluding the leading and trailing spaces,

Attribute	Type	Operation Performed
		up to (but not including) the first occurrence of any one of the delimiters; e.g.: TOKEN(" ABC / D : EF ", "/:") is "ABC".
NTOKEN (S1, I, S2)	[STR]	Returns the Ith "token" of S1, where S2 is a string containing the token delimiters; e.g. NTOKEN ('8:49/#J8/9/LDEV# FOR "T" ON TAPE (NUM)?', 3, "/") is "9".
REMTOKEN (S1, S2)	[STR]	Returns the second through last "tokens" of S1, where S2 is a string containing the token delimiters; e.g. REMTOKEN (" ABC / D : EF ", "/:") is "D : EF ".
TOKEN (S1, S2, S3)	[STR]	Same as TOKEN(S1, S2), but considers S3 to be the set of allowed QUOTE CHARACTERS – delimiters embedded within quotes are ignored in determining token boundaries. For instance, TOKEN(" 'ABC / D' : EF ", "/:", "'") is "'ABC / D'" — the first "/" is not treated as a delimiter because it's nested within "'" quote characters.
NTOKEN (S1, I, S2, S3)	[STR]	Same as NTOKEN(S1, I, S2), but supports quote characters.
REMTOKEN (S1, S2, S3)	[STR]	Same as REMTOKEN(S1, S2), but supports quote characters.

The following cases are exceptions to the syntax listed above – in them, the second parameter is not viewed as a string of delimiters, but as something else:

Attribute	Type	Operation Performed
TOKEN (S, "A")	[STR]	Returns all the characters of S until the first non-Alphabetic character. TOKEN("F001/3", "A") is "F00".
TOKEN (S, "N")	[STR]	Returns all the characters of S until the first non-Numeric character. TOKEN("123F/0", "N") is "123".
TOKEN (S, "AN")	[STR]	Returns all the characters of S until the first non-AlphaNumeric character. TOKEN("123F00/BAR", "AN") is "123F00".

Attribute	Type	Operation Performed
REMTOKEN (<i>S</i> , "A" "N" "AN")	[STR]	Returns all the characters of <i>S</i> starting with the first non-Alphabetic, non-Numeric or non-AlphaNumeric character, respectively. REMTOKEN("F001/3", "A") is "1/3".

Date operators and functions

The following operators and functions work on dates:

Attribute	Type	Operation Performed
<i>D</i> + <i>I</i>	[DATE]	Date increment; returns the date that is <i>I</i> days after <i>D</i> . E.g. (92/3/30 +10) is 92/4/9.
<i>D</i> - <i>I</i>	[DATE]	Date decrement; returns the date that is <i>I</i> days before <i>D</i> . E.g. (92/4/9 -10) is 92/3/30.
<i>D1</i> - <i>D2</i>	[INT]	Date difference; returns the number of days between <i>D1</i> and <i>D2</i> . E.g. (92/4/9 - 92/3/30) is 10.
<i>D1</i> = <i>D2</i>	[BOOL]	Tests if <i>D1</i> is the same date as <i>D2</i> .
<i>D1</i> < <i>D2</i>	[BOOL]	Tests if <i>D1</i> is earlier than <i>D2</i> .
<i>D1</i> > <i>D2</i>	[BOOL]	Tests if <i>D1</i> is later than <i>D2</i> .
<i>D1</i> <> <i>D2</i>	[BOOL]	Tests if <i>D1</i> is a different date from <i>D2</i> .
<i>D1</i> >= <i>D2</i>	[BOOL]	Tests if <i>D1</i> is later or the same as <i>D2</i> .
<i>D1</i> <= <i>D2</i>	[BOOL]	Tests if <i>D1</i> is earlier or the same as <i>D2</i> .
MIN (<i>D1</i> [, <i>D2</i> ...])	[DATE]	Returns the earliest of one (or more) dates.
MAX (<i>D1</i> [, <i>D2</i> ...])	[DATE]	Returns the latest of one (or more) dates.
BETWEEN (<i>D1</i> , <i>D2</i> , <i>D3</i>)	[BOOL]	Tests whether <i>D1</i> is between <i>D2</i> and <i>D3</i> , i.e. <i>D1</i> >= <i>D2</i> and <i>D1</i> <= <i>D3</i> .
DATEBUILD (<i>I1</i> , <i>I2</i> , <i>I3</i>)	[DATE]	Returns a date with the year <i>I1</i> , month <i>I2</i> , day <i>I3</i> ; DATEBUILD(92,10,1) is thus equal to 92/10/1. The advantage of DATEBUILD over the yy/mm/dd syntax is that DATEBUILD's parameters may be variables.
DATEYEAR (<i>D</i>)	[INT]	Returns the 2-digit year number of date <i>D</i> . Aborts if you pass it a year outside the 20th century.
DATEYEAR4 (<i>D</i>)	[INT]	Returns the 4-digit year number of date <i>D</i> .
DATEMONTH (<i>D</i>)	[INT]	Returns the month number of date <i>D</i> .

Appendix B

Attribute	Type	Operation Performed
DATEDAY (<i>D</i>)	[INT]	Returns the day-of-month of date <i>D</i> .
DATEDAYOFWEEK (<i>D</i>)	[INT]	Returns the day of week (1=Sunday, 2=Monday, 7=Saturday) of date <i>D</i> .
DATEADD (<i>D</i> , <i>I1</i> , <i>I2</i> , <i>I3</i>)	[DATE]	Returns a date that is <i>I1</i> years, <i>I2</i> months and <i>I3</i> days after the date <i>D</i> .
DAYOFYEAR (<i>D</i>)	[INT]	Returns the Julian "day of the year" for the date <i>D</i> .
YYYYMMDD (<i>D</i>)	[INT]	Converts <i>D</i> into an integer formatted as (4-digit year) followed by (2-digit month) followed by (2-digit day). YYYYMMDD(92/3/30) is 19920330.
CDATE (<i>D</i> , <i>S</i>)	[STR]	<p>[STR] Formats <i>D</i> as a string; <i>S</i> specifies format.</p> <p>CDATE(5/4/30, 'YY/MM/DD') is 05/04/30 CDATE(5/4/30, 'MM/DD/YYYY') is 04/30/2005 CDATE(5/4/30, 'DD/MM/YY') is 30/04/05</p> <p>The slashes may also be omitted or replaced by any other character, e.g.</p> <p>CDATE(5/4/30, 'YYMMDD') is 050430 CDATE(5/4/30, 'DD.MM.YY') is 30.04.05</p> <p>Other special date descriptors may also be used (see the STRWRITE function for a full description):</p> <p>CDATE(5/4/30, '%W, %1M %0D, %4Y') is SAT, APR 30, 2005</p>
CDATEYMD (<i>D</i>)	[STR]	Formats <i>D</i> as a YY/MM/DD date.
CDATEMDY (<i>D</i>)	[STR]	Formats <i>D</i> as a MM/DD/YY date.
CDATEDMY (<i>D</i>)	[STR]	Formats <i>D</i> as a DD/MM/YY date.
DATEPARSE (<i>S1</i> , <i>S2</i>)	[DATE]	<p>Parses the string <i>S1</i> as a date; <i>S2</i> specifies the format in which <i>S1</i> is supposed to be, either 'YMD', 'MDY' or 'DMY'. (The date may have either a 2- or 4-digit year and may exclude the slashes).</p> <p>DATEPARSE ('03/30/91', 'MDY') is 91/03/30; DATEPARSE ('19910330', 'YMD') is also 91/03/30.</p>
VALIDATE (<i>S1</i> , <i>S2</i>)	[BOOL]	Returns TRUE if <i>S1</i> is a valid date, FALSE if it isn't — <i>S2</i> specifies the format, either 'YMD', 'MDY' or 'DMY'. The date may have either a 2-or 4-digit year and may exclude the slashes.
TODAY	[DATE]	Returns today's date.
CALENDARTODATE (<i>I</i>)	[DATE]	Returns the date represented by the CALENDAR-format integer <i>I</i> .

Attribute	Type	Operation Performed
DATETOCALNDAR (<i>D</i>)	[INT]	Returns the CALENDAR-format representation of the date <i>D</i> .

Real operators and functions

The following operators and functions work on reals:

Note that in any "mixed-mode" operation — where one operand is an integer and the other is a real — the integer will be automatically converted to a real.

Expression or Function	Type	Operation Performed
$R1 + R2$	[REAL]	Addition.
$R1 - R2$	[REAL]	Subtraction.
$R1 * R2$	[REAL]	Multiplication.
$R1 / R2$	[REAL]	Integer division, truncates result. $20/3$ is 6.
$I1 // I2$	[REAL]	Divides two integers and yields a real number (dividing two integers as $I1/I2$ will always yield an integer); e.g. $10//3$ is 3.33333. Note that $R1/R2$ (a real divided by a real, with one slash) returns a real; two slashes are only needed when dividing integers.
ABS (<i>R</i>)	[REAL]	Absolute value. $ABS(-10) = 10$; $ABS(10) = 10$.
MIN (<i>R1</i> [, <i>R2</i> ...])	[REAL]	Returns the smallest of one (or more) reals.
MAX (<i>R1</i> [, <i>R2</i> ...])	[REAL]	Returns the largest of one (or more) reals.
$R1 ^ R2$	[REAL]	" <i>R1</i> to the power <i>R2</i> ". E.g. 2.5^3 is 15.625, $2.25^.5$ is 1.5.
$R1 = R2$	[BOOL]	Tests whether <i>R1</i> is equal to <i>R2</i> .
$R1 < R2$	[BOOL]	Tests whether <i>R1</i> is less than <i>R2</i> .
$R1 > R2$	[BOOL]	Tests whether <i>R1</i> is greater than <i>R2</i> .
$R1 <> R2$	[BOOL]	Tests whether <i>R1</i> is not equal to <i>R2</i> .
$R1 >= R2$	[BOOL]	Tests whether <i>R1</i> is greater than or equal to <i>R2</i> .
$R1 <= R2$	[BOOL]	Tests whether <i>R1</i> is less than or equal to <i>R2</i> .
BETWEEN (<i>R1</i> , <i>R2</i> , <i>R3</i>)	[BOOL]	Tests whether <i>R1</i> is between <i>R2</i> and <i>R3</i> , i.e. $R1 >= R2$ AND $R1 <= R3$.
ROUND (<i>R</i>)	[INT]	Rounds <i>R</i> to the nearest integer. E.g. $ROUND(1.5)$ is 2.0.

Expression or Function	Type	Operation Performed
INTEGER (<i>R</i>)	[INT]	Truncates <i>R</i> into an integer.
REALPARSE (<i>S</i>)	[REAL]	Converts the string <i>S</i> to a real number; if it's not a valid real number, aborts with an error message.
EXP (<i>R</i>)	[REAL]	Returns e (2.718281828) to the power <i>R</i> .
LOG (<i>R</i>)	[REAL]	Returns the natural logarithm of <i>R</i> .
SIN (<i>R</i>)	[REAL]	Returns the sine of <i>R</i> (an angle in radians).
COS (<i>R</i>)	[REAL]	Returns the cosine of <i>R</i> (an angle in radians).
TAN (<i>R</i>)	[REAL]	Returns the tangent of <i>R</i> (an angle in radians).
SQRT (<i>R</i>)	[REAL]	Returns the square root of <i>R</i> .
REAL (<i>I</i>)	[REAL]	Explicitly converts <i>I</i> to a real.

Time operators and functions

Constants of type time may be specified in one of three formats:

```
hour:minuteAM      e.g. 10:25AM
hour:minutePM      e.g.  6:45PM
hour:minuteT       e.g. 17:07T      (24-hour clock)
```

For example, you might say (the function **CLOCK** is described below):

```
CLOCK>6:00PM AND CLOCK<6:30PM
```

or

```
CLOCK>18:00T AND CLOCK<18:30T
```

If the "minute" field is 0 (e.g. 6:00PM), you can omit it and the colon, i.e. say "6PM"; for instance,

```
CLOCK>6PM AND CLOCK<7PM
```

You must specify one of those three suffixes (to avoid ambiguity) — "9:55" is not valid (though "9:55T" is).

The following operators and functions work on time values:

Expression or Function	Type	Operation Performed
<i>T</i> + <i>I</i>	[TIME]	Returns the time that is <i>I</i> seconds after time <i>T</i> .
<i>T</i> - <i>I</i>	[TIME]	Returns the time that is <i>I</i> seconds before time <i>T</i> .
<i>T1</i> - <i>T2</i>	[INT]	Returns the number of seconds between <i>T2</i> and <i>T1</i> .

Expression or Function	Type	Operation Performed
MIN (<i>T1</i> [, <i>T2</i> ...])	[TIME]	Returns the earliest of one (or more) times.
MAX (<i>T1</i> [, <i>T2</i> ...])	[TIME]	Returns the latest of one (or more) times.
<i>T1</i> = <i>T2</i>	[BOOL]	Tests whether <i>T1</i> is equal to <i>T2</i> .
<i>T1</i> < <i>T2</i>	[BOOL]	Tests whether <i>T1</i> is less than <i>T2</i> .
<i>T1</i> > <i>T2</i>	[BOOL]	Tests whether <i>T1</i> is greater than <i>T2</i> .
<i>T1</i> <> <i>T2</i>	[BOOL]	Tests whether <i>T1</i> is not equal to <i>T2</i> .
<i>T1</i> >= <i>T2</i>	[BOOL]	Tests whether <i>T1</i> is greater than or equal to <i>T2</i> .
<i>T1</i> <= <i>T2</i>	[BOOL]	Tests whether <i>T1</i> is less than or equal to <i>T2</i> .
BETWEEN (<i>T1</i> , <i>T2</i> , <i>T3</i>)	[BOOL]	TRUE if <i>T1</i> is between <i>T2</i> and <i>T3</i> (inclusively); BETWEEN(CLOCK,8AM,8:15AM) returns TRUE if CLOCK>=8AM and CLOCK<=8:15AM. Note: always FALSE if <i>T2</i> > <i>T3</i> ; i.e. BETWEEN(CLOCK,10PM,4AM) is always false — instead use NOT BETWEEN(CLOCK,4AM,10PM).
CLOCK	[TIME]	Returns the current time.
CLOCKTotime (<i>I</i>)	[TIME]	Converts an integer in the format returned by the MPE CLOCK intrinsic to the VESoft time format.
HMMSS (<i>T</i>)	[INT]	Formats the time <i>T</i> into an integer consisting of (2- digit hour) followed by (2-digit minute) followed by (2-digit second); HMMSS(5:33PM) is 173300.
TIMEPARSE (<i>S</i>)	[TIME]	Parses the string <i>S</i> as a time; TIMEPARSE('23:00') is 11:00 PM; TIMEPARSE('23:00T') is also 11:00 PM; TIMEPARSE('11:00') is 11:00 AM, not PM; TIMEPARSE('11:00 PM') is 11:00 PM.
VALIDTIME (<i>S</i>)	[BOOL]	Returns TRUE if the string <i>S</i> contains a valid time representation (e.g. "12:34 PM"), FALSE if it doesn't.
TIMEBUILD (<i>I1</i> , <i>I2</i> , <i>I3</i>)	[TIME]	Constructs a time value corresponding to <i>I1</i> hours, <i>I2</i> minutes and <i>I3</i> seconds. Thus, TIMEBUILD(10,5,0) is the same as 10:05AM —TIMEBUILD's advantage over the hh:mmT syntax is that TIMEBUILD can take variables as parameters.

Special MPE access functions

These functions do special MPE-dependent things like look up your capabilities, parse/format file codes, etc.:

Expression or Function	Type	Operation Performed
CAPABILITY (<i>S</i>)	[BOOL]	Checks if the running user has the capability indicated by the 2-character string <i>S</i> , e.g. CAPABILITY("SM") checks if the user has SM capability.
ICODE (<i>S</i>)	[INT]	Parses a string that contains an MPE filecode and returns the integer value of the filecode, e.g. ICODE("PROG") is 1029. Remember, "PRIV" and "KSAM" are NOT filecodes (even though :LISTF, 2 puts them in the CODE field).
SCODE (<i>I</i>)	[STR]	Converts an integer value of an MPE filecode into its string representation, e.g. SCODE(1029) is "PROG".
MPE (<i>S</i>)	[INT]	Executes the MPE command indicated by <i>S</i> ; returns 0 if all's well or the CI error number if an error does occur
WRITEPEMESSAGE (<i>I1</i> , <i>I2</i>)	[NONE]	Outputs the MPE error message <i>I2</i> in message set <i>I1</i> . CIERRs are in message set 2, FSERRs are in message set 8. E.g. WRITEPEMESSAGE(2,980) writes "COMMAND GREATER THAN 278 CHARACTERS LONG. (CIERR 980)" to your terminal; WRITEPEMESSAGE(8,52) would write "NONEXISTENT PERMANENT FILE (FSERR 52)".
WRITEPEXLSTATUS (<i>I</i>)	[NONE]	Outputs the message corresponding to the MPE/iX intrinsic status code passed as <i>I</i> .
INPUT ([<i>S</i>] [, <i>I</i>])	[STR]	Prompts the user with string <i>S</i> (optional), then waits <i>I</i> seconds (optional, default forever) for a response. Returns the response as a string. If the user does not respond before the timeout, "" (the null string) is returned and a warning is issued.
PRINTOPREPLY (<i>S</i>)	[STR]	Sends the string <i>S</i> as a message to the system console and waits for the operator to :REPLY. Returns the value given by the operator in the :REPLY command.
SONALIVE (<i>I</i>)	[BOOL]	Returns TRUE if the process whose PIN is <i>I</i> is alive and a son process of MPEX. Useful for checking on

Expression or Function	Type	Operation Performed
		the progress of son processes that you've started (whose PINs you got from the MPEXPIN JCW).
SONALIVENAME (<i>I</i> , <i>S</i>)	[BOOL]	Returns TRUE if the process whose PIN is <i>I</i> is alive, is a son process of MPEX and is running the program <i>S</i> .
ISCHLOGONED ()	[BOOL]	Returns TRUE if session has done a CHLOGON to another logon (and hasn't switched back yet).
BOUND (<i>varname</i>)	[BOOL]	True if the variable <i>varname</i> exists. e.g. BOUND(MYVAR) returns true if the variable "MYVAR" exists. Note: do not put quotes around the variable name.
TYPEOF (<i>expression</i>)	[INT]	Returns the MPE type of an expression or variable, or returns an error if the expression is invalid. This function returns: 1 if expression evaluates to an INTEGER 2 if expression evaluates to a STRING 3 if expression evaluates to a BOOLEAN 101 if expression evaluates to a REAL number 102 if expression evaluates to a DATE 103 if expression evaluates to a TIME Note: when MPEX expressions of type DATE, TIME and REAL are assigned to a variable, the variable will be of type STRING.
SVAR (<i>S</i>)	[STR]	Retrieves the value of an MPE variable (either predefined [see below], set by the MPEX %SETVAR command or a JCW) as a string; if the variable is not a string variable, formats it as a string. If the variable does not exist, returns an empty string. For instance, SVAR("HPACCOUNT") returns the value of the predefined HPACCOUNT variable — the logon account.
SVAR (<i>S1</i> , <i>S2</i>)	[STR]	Same as SVAR(<i>S</i>), but returns the string <i>S2</i> if the variable <i>S1</i> doesn't exist.
IVAR (<i>S</i>)	[INT]	Retrieves the value of an MPE variable as an integer; if the variable is a string or does not exist, returns 0.
IVAR (<i>S</i> , <i>I</i>)	[INT]	Same as IVAR(<i>S</i>), but returns the integer <i>I</i> if the variable is a string or doesn't exist.
BVAR (<i>S</i>)	[BOOL]	Retrieves the value of an MPE variable as a boolean; if the variable does not exist, returns FALSE.

Appendix B

Expression or Function	Type	Operation Performed
BVAR (<i>S</i> , <i>B</i>)	[BOOL]	Same as BVAR(<i>S</i>), but returns the value <i>B</i> if the variable doesn't exist.
SVARSET (<i>S1</i> , <i>S2</i>)	[STR]	Sets the variable <i>S1</i> to the string value <i>S2</i> . Returns the value of <i>S2</i> .
IVARSET (<i>S</i> , <i>I</i>)	[INT]	Sets the variable <i>S</i> to the integer value <i>I</i> . Returns the value of <i>I</i> .
BVARSET (<i>S</i> , <i>B</i>)	[BOOL]	Sets the variable <i>S</i> to the boolean value <i>B</i> . Returns the value of <i>B</i> .
SETVAR (<i>S</i> , <i>expression</i>)	[???	<p>Sets the variable <i>S</i> to the value of "<i>expression</i>". Returns the value of "<i>expression</i>".</p> <p>The type depends on the type of "<i>expression</i>" (integer, string or boolean).</p> <p>SETVAR(TESTVAR, "TESTVALUE") returns "TESTVALUE" and sets the variable TESTVAR to "TESTVALUE".</p>
CMDPARM (<i>S</i>)	[STR]	<p>Returns the value of the parameter indicated by <i>S</i> of the currently-executing command file or UDC.</p> <p>Why use CMDPARM("XYZ") and not "!XYZ"?</p> <p>Because CMDPARM works even when the parameter XYZ contains embedded quotes — the "!XYZ" syntax would cause a syntax error in this case.</p>
RANDOMNAME	[STR]	Returns a string of 8 random alphabetic characters, useful for setting random passwords and lockwords. The letters are alternating consonants and vowels (for ease of pronouncing and remembering). There are over 100 million possible values of this function.
PAUSE (<i>I</i>)	[NONE]	Pauses for <i>I</i> seconds. Returns no result; therefore, it can only be used in a %CALC PAUSE(<i>I</i>) or in an "expression program".
ACCTEXISTS (<i>S</i>)	[BOOL]	TRUE if account indicated by the string <i>S</i> exists. E.g. ACCTEXISTS("SYS") is TRUE.
USEREXISTS (<i>S</i>)	[BOOL]	<p>TRUE if the user.account indicated by the string <i>S</i> exists. E.g.</p> <p>USEREXISTS("MANAGER.SYS") is TRUE.</p>
GROUPEXISTS (<i>S</i>)	[BOOL]	<p>TRUE if the group.account indicated by the string <i>S</i> exists. E.g.</p> <p>GROUPEXISTS("PUB.SYS") is TRUE.</p>

Expression or Function	Type	Operation Performed
VEACCTINFO (<i>S</i>). <i>attrvar</i>	[???	Returns the MPE account attribute variable indicated by " <i>attrvar</i> " for the account indicated by <i>S</i> . " <i>attrvar</i> " must be one of the account attribute variables discussed in the "Object Attribute Variables" chapter of the VEAUDIT User Manual.
VEUSERINFO (<i>S</i>). <i>attrvar</i>	[???	Returns the MPE user attribute variable indicated by " <i>attrvar</i> " for the user.account indicated by <i>S</i> . " <i>attrvar</i> " must be one of the user attribute variables discussed in the "Object Attribute Variables" chapter of the VEAUDIT User Manual.
VEGROUPINFO (<i>S</i>). <i>attrvar</i>	[???	Returns the MPE group attribute variable indicated by " <i>attrvar</i> " for the group.account indicated by <i>S</i> . " <i>attrvar</i> " must be one of the group attribute variables discussed in the "Object Attribute Variables" chapter of the VEAUDIT User Manual.
VEJOBINFO (<i>jobtype</i> , <i>jobnum</i>). <i>attrvar</i>	[???	Returns the job attribute variable indicated by " <i>attrvar</i> " for the given job. " <i>Jobtype</i> " is a string and must be either "J" or "S"; " <i>jobnum</i> " must be an integer expression. " <i>attrvar</i> " must be one of the job/session attribute variables discussed in the "Job/Session Attributes" chapter of the MPEX User Manual.
VEJOBINFO (<i>jobid</i>). <i>attrvar</i>	[???	Same as above, but " <i>jobid</i> " is a string of the format '#J999' or '#S999'.
JSCOUNT (<i>S</i>)	[INT]	Counts the number of jobs or sessions that match the user set given by the string <i>S</i> ; for instance, JSCOUNT("@.AP-MGR.AP&ONLINE") returns the number of sessions ("&ONLINE") that are logged on to the AP account ("@.AP") but not as MGR.AP ("MGR.AP"). Note that the evaluation of +, - and & proceeds from left to right (unless you use parentheses to override this). "User sets" are discussed in more detail in the SECURITY User Manual.

HP terminal manipulation functions

The following functions make it easier for you to do special HP terminal-dependent things (e.g. position the cursor, clear the screen, etc.) — useful in command files, SECURITY menus, etc.:

Expression or Function	Type	Operation Performed
HPTERMINAL ()	[BOOL]	Returns TRUE if the terminal is configured as an HP terminal (terminal type 10), FALSE if it isn't. Note that this function must be called as HPTERMINAL () — you can not omit the "()".
TERMPPOS (<i>I1</i> , <i>I2</i>)	[STR]	Returns the escape sequence that causes an HP terminal to position its cursor at row <i>I1</i> , column <i>I2</i> (counting from 0).
TERMCLEAR ()	[STR]	Returns the escape sequence that causes an HP terminal to clear the rest of the screen from the cursor to the end of memory. To clear the entire screen, you should use TERMPPOS (0,0)+TERMCLEAR (). Note that this function must be called as TERMCLEAR () — you can not omit the "()".
TERMFKEY (<i>I</i>)	[STR]	Returns the string that corresponds to hitting the function key numbered <i>I</i> on an HP terminal. Used to check if a user hit a particular function key, e.g. IF USERINPUT=TERMFKEY(8) THEN ... <i>I</i> must be between 1 and 8.
TERMFKEYNUMBER (<i>S</i>)	[INT]	If <i>S</i> is the string that is sent by an HP terminal when a function key is hit, returns the number (1 through 8) of that function key; if it isn't, returns 0. Useful for checking to see if a user hit a function key and if so, which one.

SECURITY-related functions

The following functions let you do special SECURITY-related things:

Expression or Function	Type	Operation Performed
VEPROFILEEXISTS (<i>S</i>)	[BOOL]	Returns TRUE if SECURITY/3000 user profile for the session,user.account indicated by the string <i>S</i> exists; e.g. VEPROFILEEXISTS("NIVEN,WRITER.SF")
VEPROFILEINFO (<i>S</i>). <i>attrvar</i> [???]		Returns the SECURITY/3000 user profile attribute variable indicated by " <i>attrvar</i> " for the session,user.account indicated by <i>S</i> . " <i>Attrvar</i> " must be one of the user profile attribute variables discussed in the "Listing User Profiles" chapter of the SECURITY User Manual. For example, VEPROFILEINFO("LN,WRITER.SF").NAME might be the string "Larry Niven".

SECURITYLOG (<i>S</i>)	[NONE]	Writes a record containing the given string <i>S</i> to the SECURITY log file. This record will then show up (with type "Manual") in the SECURITY LISTLOG entry point output or can be specially looked for using the OPMANUAL and QUALIFIER selection conditions. Returns no result; therefore, it can only be used in a %CALC SECURITYLOG(...) or in an "expression program".
BADPASSWORD (<i>S</i>)	[BOOL]	Returns TRUE if the string <i>S</i> is one of the "bad passwords" in the BADPASS.AUDITDAT.VESOFT list of easy to guess passwords (see the VEAUDIT User Manual), and prints the associated error message; otherwise returns FALSE.

File attribute functions

These functions let you do some pretty interesting things to files. Don't let their position near the end of the list fool you — these are some of the most useful expressions we provide.

Note: MPEX's FINFO, VEFINFO and FEXISTS functions normally return information about permanent files; MPE/iX's FINFO function, however, first looks for the file as a temporary file and only then looks at the permanent file.

By doing a

```
:SETJCW MPEXFINFOANY=1
```

you can make MPEX's FINFO, VEFINFO and FEXISTS functions behave like MPE/iX's FINFO; i.e. they will look first for a temporary file and then (if no temporary file with the given name exists) for a permanent file.

Expression or Function	Type	Operation Performed
FREMLOCKWORD (<i>S</i>)	[STR]	Takes a string <i>S</i> that contains a filename and returns the same filename, fully qualified but WITHOUT a lockword (if any).
OBJECTFILE (<i>S</i>)	[STR]	(Only usable within MPEX's %REPEAT...%FORFILES construct, documented in the MPEX User Manual.) Returns the name of a file in the "object fileset" indicated by <i>S</i> that corresponds to the current file in the "source fileset". For example, if the "source fileset" is S@.DATA and the current filename is SMYPROG.DATA, then OBJECTFILE("B@.BACKUP") is "BMYPROG.BACKUP".

Expression or Function	Type	Operation Performed
FEXISTS (<i>S</i>)	[BOOL]	Checks to see if the permanent file indicated by <i>S</i> exists — if it does, returns TRUE; if it doesn't, FALSE. Set the JCW MPEXFINFOANY to 1 to make FEXISTS look first for a temp file and then, if it doesn't find one, for a perm file.
FTEMP EXISTS (<i>S</i>)	[BOOL]	Just like FEXISTS, but for temp files.
FPERM EXISTS (<i>S</i>)	[BOOL]	Just like FEXISTS; use this function if you SETJCW MPEXFINFOANY=1 and you want to only check for a permanent file.
FANY EXISTS (<i>S</i>)	[BOOL]	True if either a permanent or a temp file with the given name exists.
VEFINFO (<i>S</i>). <i>attrvar</i>	[???	<p>Finds the permanent file <i>S</i> and returns an arbitrary piece of information about the file indicated by "<i>attrvar</i>".</p> <p>"<i>attrvar</i>" must be one of the file attribute variables discussed in "File Attribute Variables And Functions" in the MPEX Filesets chapter of the MPEX User Manual — but not PROG ... or SPOOL ...</p> <p>For example:</p> <p>VEFINFO("AP010S.DATA").CODE returns AP010S.DATA's filecode as a string</p> <p>VEFINFO("X.PUB").SECTORS returns the number of sectors used by X.PUB;</p> <p>VEFINFO(DBSET+"DB").DBSETNAME returns the IMAGE dataset name of the dataset whose file name is in the DBSET variable in the DB group.</p> <p>Set the JCW MPEXFINFOANY to 1 to make VEFINFO look first for a temp file and then, if it doesn't find one, for a perm file.</p>
VEFTEMPINFO (<i>S</i>). <i>attrvar</i>	[???	Just like VEFINFO, but for temp files.
VEFPERMINFO (<i>S</i>). <i>attrvar</i>	[???	Just like VEFINFO; use this function if you SETJCW MPEXFINFOANY=1 and you want to only check for a permanent file.
VEFANYINFO (<i>S</i>). <i>attrvar</i>	[???	Just like VEFINFO, but check first for a temp file, then a permanent file.
FINFO (<i>S</i> , <i>I</i>)	[???	The FINFO function allows you to extract various pieces of information about a (by default, permanent) file named by <i>S</i> . The information you get — and its type — depends on the value of the second parameter (<i>I</i>), which must be an integer constant. Note that all

Expression or Function	Type	Operation Performed
		string results have trailing blanks stripped. If the file doesn't exist, an error is reported.
		Although MPE/iX's FINFO function looks first for a temp file and then, if it doesn't find one, for a perm file, the MPEX FINFO function only looks for a perm file. You can set the JCW MPEXFINFOANY to 1 to make FINFO work the same way in MPEX that it does in MPE/iX.
FTEMPINFO (<i>S</i> , <i>I</i>)	[???	Just like FINFO , but only looks at temp files.
FPERMINFO (<i>S</i> , <i>I</i>)	[???	Just like FINFO , but only looks at permanent files.
FANYINFO (<i>S</i> , <i>I</i>)	[???	Just like FINFO , but looks first for a temp file and then (if it doesn't find one), for a permanent file.

FINFO Function Attributes

Both MPE/iX and MPEX now allow a string to be supplied as the second parameter of the **FINFO** function, instead of the hard-to-remember numeric parameter originally supported.

As the list of supported string attributes is poorly documented by HP, we include the complete list here:

Attribute	#	Attribute	#
EXISTS	0	SECURITY MATRIX	-5
FULL FILENAME	1	CREATED	6
FULLFNAME	1	CREATION DATE	6
FULLY QUALIFIED FILENAME	1	FMTCREATED	6
FILENAME ONLY	-1	CREATION DATE INTEGER	-6
FNAME	-1	INTCREATED	-6
GROUP NAME	2	ACCESSED	7
GROUP	2	FMTACCESSED	7
ACCOUNT NAME	3	LAST ACCESS DATE	7
ACCOUNT	3	INTACCESSED	-7
ACCT	3	LAST ACCESS DATE INTEGER	-7
CREATOR	4	FMTMODDATE	8
FMTSECURITY	5	LAST MOD DATE	8
FORMATTED SECURITY MATRIX	5	MODIFIED	8
INTSECURITY	-5	INTMODDATE	-8

Appendix B

Attribute	#
LAST MOD DATE INTEGER	-8
FILE CODE MNEMONIC	9
FMTFCODE	9
FCODE	-9
FILE CODE	-9
INTFCODE	-9
USER LABELS WRITTEN	10
USER LABELS AVAIL	11
FILE LIMIT	12
LIMIT	12
FMTFOPT	13
FORMATTED FOPTIONS	13
FOPTIONS	-13
INTFOPT	-13
RECORD SIZE	14
RECSIZE	14
BLOCK SIZE	15
BLKSIZE	15
MAX EXTENTS	16
MAXEXT	16
LAST EXTENT SIZE	17
LASTTEXTSIZE	17
EXTENT SIZE	18
EXTSIZE	18
END OF FILE	19
EOF	19
ALLOC TIME	20
FMTALLOCTIME	20
ALLOC TIME INTEGER	-20
INTALLOCTIME	-20
ALLOC DATE	21

Attribute	#
ALLOCATED	21
FMTALLOCDATE	21
ALLOC DATE INTEGER	-21
INTALLOCDATE	-21
NUM OPEN CLOSE RECS	22
DEVICE NAME	23
DEVNAME	23
FMTMODTIME	24
LAST MOD TIME	24
INTMODTIME	-24
LAST MOD TIME INTEGER	-24
FIRST USER LABEL	25
UFID	27
UNIQUE FILE ID	27
BYTE FILE LIMIT	28
BYTELIMIT	28
BYTE DATA OFFSET	29
DATASTART	29
BYTE RECORD SIZE	30
BYTERECSIZE	30
BYTE BLOCK SIZE	31
BYTEBLKSIZE	31
BYTE EXTENT SIZE	32
BYTEEXTSIZE	32
LOCKWORD	33
VOLRESTR	34
VOLUME RESTRICTION	34
The following attributes are only available on POSIX systems:	
VOLUME SET NAME	35
LOG SET ID	36

Attribute	#
LOGICAL DEVICE NUMBER	37
LDEV	37
POSIXFULLFNAME	38
POSIX FULL FILENAME	38
NUM HARD LINKS	39
NUMHARDLINKS	39
LAST ACCESS TIME	40
ACCESS TIME	40
FMTACCESSTIME	40
LAST ACCESS TIME INTEGER	-40
INTACCESSTIME	-40
STATUS CHANGE TIME	41
FMTSTATUSCHANGETIME	41
INTSTATUSCHANGETIME	-41
STATUS CHANGE DATE	42
CHANGE DATE INTEGER	-42
OWNER	43
FILE OWNER NAME	43
UID	44
FILE OWNER ID	44
FILE GROUP	45
FILE GROUP NAME	45
GID	46
FILE GROUP ID	46
FILETYPE	47
FILE TYPE	47
FILE TYPE INTEGER	-47
INTFILETYPE	-47

Attribute	#
RECTYPE	48
RECORD TYPE	48
BYTEFILESIZE	49
BYTE FILE SIZE	49
KSAM VERSION	50
KSAMVERS	50
KSAMPARAM	51
KSAM LABEL	51
DEVTYPE	52
DEVICE TYPE	52
DEVICE TYPE INTEGER	-52
INTDEVTYPE	-52
RELEASED	53
COMPRESSED	56
MIGRATED	57
SECTORS	58
NUM SECTORS	58
EXTENTS	59
NUM EXTENTS	59
CREATETIME	60
FMTCREATETIME	60
CREATION TIME	60
INTCREATETIME	-60
CREATION TIME INTEGER	-60
ACCESSORS	61
NUM ACCESSORS	61
LARGE	64

Note that many of these entries are synonyms and that negative numeric parameters return the same information as corresponding positive ones, but unformatted.

FILE I/O functions

With these functions, you can access (open, read, write, close, etc.) disc files, including KSAM files and IMAGE databases (see below), from within MPEX, STREAMX, VEMENU, SECURITY's \$FORBID, etc. — anywhere that you can use the expression handler. (In MPEX, you'll probably use them from the %SETVAR, %CALC and perhaps %WHILE and %IF commands.)

Collectively, these routines are referred to as "VEFxxx functions" in our documentation.

It is important to note that SM users may use these functions to open, read and modify PRIV files. In order to open a PRIV file, the file parameter of the VEFOPEN and VEFREAD functions must include the correct file code.

For instance, to open the ROOT file of a database directly, you would use:

```
SETVAR ROOTFILE VEFOPEN('MYDB,OLD;CODE=-400');
```

Warning: PRIV files are privileged for a reason! Writing to PRIV files, such as IMAGE DATABASE files, may corrupt the internal structure of the file and make it useless to your application. Use this feature with care — VESOFT, Inc. cannot repair files corrupted (whether PRIVileged or not) by misuse of these functions.

These functions roughly correspond to the file system intrinsics (see HP's MPE INTRINSICS REFERENCE MANUAL for details), but are often more powerful and easier to use. The functions are:

Function	Type	File System Call
VEFOPEN (<i>'filename, domain, fileeqparms'</i>)	[INT]	<p>Returns an integer file number to be used by the other functions. The string you pass to VEFOPEN should look very much like a :FILE equation (without the word "FILE"), e.g.</p> <p>VEFOPEN('MYFILE,OLD;ACC=INOUT;SHR').</p> <p>However, several new parameters are supported:</p> <p>;OLDANY try to open this as a temporary file, and if that fails, as a permanent one.</p> <p>;NOFILEEQ ignore :FILE equations.</p> <p>;ULABEL=<i>num</i> create the new file with room for up to <i>num</i> user labels.</p> <p>;BUILD build the file first, and then re-open it.</p> <p>;MAYBUILD if the file already exists, simply open it; if it doesn't exist, build it first.</p> <p>You must specify ",OLD" when using ";MAYBUILD".</p>
VEFCLOSE (<i>fnum[,disposition]</i>)	[NONE]	<p>Closes the given file number. "<i>disposition</i>" should be a string and may contain the following keywords delimited by semicolons (e.g. "SAVE;SQUEEZE"):</p>

Function	Type	File System Call
		SAVE save file as permanent. TEMP save file as temporary. DEL purge file. SQUEEZE set file limit to be equal to EOF. XLTRIM return to free space any unused space beyond EOF (MPE/iX only). CREATORONLY save file with access mask set to allow access only to its creator. If <i>disposition</i> is omitted, the file will be closed without any of the above parameters (which means leave the file as it is, just like FCLOSE disposition 0).
VEFREAD (<i>FNUM</i>)	[STR]	Reads the next record from the file; sets VEEOF to TRUE if read encounters end-of-file.
VEFREADV (<i>S</i> , <i>FNUM</i>)	[BOOL]	Reads the next record from the file and puts it into the variable given by <i>S</i> ; returns TRUE if all OK, FALSE if EOF (also sets VEEOF to TRUE if read encounters end-of-file). Example: WHILE VEFREADV("REC",MYFNUM) DO ... loops through the records in MYFNUM, putting each one in turn into the variable REC.
VEFREADDIR (<i>FNUM</i> , <i>I</i>)	[STR]	Reads record <i>I</i> from the file. May only be used with fixed record-length files.
VEFREADLABEL (<i>FNUM</i> , <i>I</i>)	[STR]	Reads label <i>I</i> from the file.
VEFWRITE (<i>FNUM</i> , <i>S</i> [, <i>I</i>])	[NONE]	Writes <i>S</i> to the file, optionally with carriage control <i>I</i> .
VEFWRITEDIR (<i>FNUM</i> , <i>S</i> , <i>I</i>)	[NONE]	Writes <i>S</i> as record <i>I</i> to the file.
VEFWRITELABEL (<i>FNUM</i> , <i>S</i> , <i>I</i>)	[NONE]	Writes <i>S</i> as label <i>I</i> to the file.
VEFPOINT (<i>FNUM</i> , <i>I</i>)	[NONE]	Points the file to record <i>I</i> .
VEFCONTROL (<i>FNUM</i> , <i>I</i> , <i>J</i>)	[INT]	Does an FCONTROL mode <i>I</i> with parameter <i>J</i> . Returns the value returned by FCONTROL's third Parameter.
VEFLOCK (<i>FNUM</i> , <i>L</i>)	[NONE]	Locks the file; if <i>L</i> =TRUE and file already locked by someone else, waits; if <i>L</i> =FALSE, doesn't wait.
VEFUNLOCK (<i>FNUM</i>)	[NONE]	Unlocks the file.
VEFINFO (<i>FNUM</i>).xxx	[???	Returns information about the file — just like a normal VEFINFO, but of an open file.
VEPRINTFILEINFO (<i>FNUM</i>)	[NONE]	Prints a "tombstone" containing various information about the file.

Function	Type	File System Call
VEFREADBYKEY (<i>FNUM</i> , <i>S</i> , <i>I</i>)	[STR]	Reads a KSAM file record with key <i>S</i> (key start position given as <i>I</i>); sets VEEOF to TRUE if the record was not found.
VEFREADBYKEYV (<i>S1</i> , <i>FNUM</i> , <i>S2</i> , <i>I</i>)	[BOOL]	Analogous to VEFREADV.
VEFFINDBYKEY (<i>FNUM</i> , <i>S</i> , <i>I</i> , <i>S1</i>)	[BOOL]	Finds a KSAM file record with key equal to <i>S</i> (if <i>S1</i> is "="), greater than <i>S</i> (if <i>S1</i> is ">") or greater than or equal to <i>S</i> (if <i>S1</i> is ">="). Key start position given as <i>I</i> ; partial key lookup is supported (if <i>S</i> is shorter than the key length). This function returns TRUE if found or FALSE if not found and sets the VEEOF variable to the opposite setting (i.e., FALSE if found and TRUE if not found).
VEFREMOVE (<i>FNUM</i>)	[STR]	Removes the current record in the KSAM file.
VEFUPDATE (<i>FNUM</i> , <i>S</i>)	[STR]	Replaces the current record in the KSAM file by <i>S</i> .
VEFFINDN (<i>FNUM</i> , <i>I</i> , <i>J</i>)	[STR]	Finds record number <i>I</i> (key start position given as <i>J</i>).
VEFREADC (<i>FNUM</i>)	[STR]	Reads the chronologically next record.
VEFREADCv (<i>S</i> , <i>FNUM</i>)	[BOOL]	Analogous to VEFREADV.

- If there is any error accessing the file, the command will abort with **CIERROR** set to 30080 and **FSERROR** set to the file system error number; you can, of course, use %CONTINUE, %IF, %TRAPERROR, etc. to do appropriate error handling. The only exception to this are an end of file discovered by VEFREAD or VEFREADC (not by any of the other VEFREADxxx functions) or a nonexistent key discovered by VEFREADBYKEY or VEFFINDBYKEY; these will cause no error, but rather return an empty string and set the special VEEOF variable to TRUE. The alternate VEFREADV, VEFREADCv and VEFREADBYKEYV functions return the eof/no key condition as their result and put the record that was read into a variable whose name is passed to them.
- If any of the file I/O functions encounter a file system error, the variable VEFSEERR is set to the MPE FSERROR number.
- The VEFREADxxx and VEFWRITExxx functions can be passed a filename/file-parameter string (like the one that you can pass to VEFOPEN) as the first parameter, instead of the file number. We will then open the file, do the operation and immediately close the file; thus,

```
VEFREADDIR ( 'MYFILE,OLD', 5)
```

will return record 5 of the file MYFILE.

It is important that you specify ,OLD;ACC=OUTKEEP (for VEFWRITExxx calls) and whatever other parameters you need to make the I/O work properly.

- The VEFREADxxxV functions return the record that is read into an MPEX (or MPE/iX) variable, not an expression program variable created by the VAR statement in an expression program (see "[Advanced expression programming](#)" below).

- What if the file you're reading or writing contains binary (rather than ASCII) data? You can extract this data from the record by using the new `INTEGER` function, e.g.

```
INTEGER(REC[10:2])
```

to extract the 2-byte integer starting with byte 10 (counting from 0) of `REC`; you can assemble a record from ASCII and binary data by using the `'I2'` and `'I4'` format specifiers of `STRWRITE`, e.g.

```
VEFWRITE (MYFNUM, STRWRITE(NAME:30, NUMKIDS:'I2', SSN:'I4'))
```

For more documentation on `INTEGER`, see "[Integer functions](#)"; for `'I2'` and `'I4'`, see "[Output formatting](#)" below.

This facility has certain limitations that you should be aware of. Some of them may be removed in the future, but we aren't making any promises.

- These functions are probably not fast enough to do serious processing of large files. Actually, it's not so much the functions that are slow as the `MPEX` commands — like `%CALC` and `%SETVAR` — in which you'll use the functions.
If you want to process a few records of a very large file, you might want to find them first using the comparatively fast `%PRINT...`; `SEARCH=`, send its output to a temporary file and then process the file with the `VEFxxx` functions.
- Because string variables can be at most 255 characters long, you'll have trouble processing files with records longer than 255 characters.
- `VEFINFO` currently works only on disc, non-remote files.

Here is an example making use of some of these functions. It uses a file named `STRLIST`, which is a list of strings to search for and reports all the files in a fileset that contain ANY of those strings.

```

PARM CS$FILESET
REPEAT
SETVAR FNUM VEFOPEN("STRLIST,OLD")
TRAPERROR
WHILE VEFREADV("CURREC",FNUM) DO
    SETVAR SEARCHSTR TOKEN(CURREC," ")
    IF RFILE.FSEARCHSTRING(SEARCHSTR)>0 THEN
        ECHO !MPEXCURRENTFILE contains the string !SEARCHSTR
    ENDIF
ENDWHILE
CLEANUP
    CALC VEFCLOSE(FNUM)
    DELETEVAR CURREC
    DELETEVAR FNUM
ENDCLEANUP
FORFILES !FILESET
```

Database I/O functions

These functions provide access to IMAGE databases and are similar to the File I/O functions above. Please don't throw away your applications now that you have these functions. You won't get incredibly fast performance from them, but if you only need to affect an entry or two, they can be extremely convenient!

Opening a database for access

Note that in the function definitions shown below, *dbid* can be either the string returned by VEDBOPEN or a string containing '*dbname,pass,mode*' (in the latter case, the open and close will be done automatically). *dset* is a string containing the name of the dataset being accessed. *list* is a string containing an ordered list of data item names (delimited by commas) or '@' to indicate all items. *buffer* is a string containing the data item values specified in the *list* parameter (and in the same order).

If you are doing a single operation, such as adding a new record to a detail set, you should use the automatic method of opening, locking, and closing a database. To do this, simply specify the name of the database, the password, and the mode to open the database as a string constant or variable in the call to VEDBPUT. If you intend to perform a number of operations, such as finding a detail record, updating a few fields, then re-writing the detail record, it may be better to use the VEDBOPEN function to keep the database open between VEDB calls.

In either case, these functions will look for and evaluate a database Access Control File if one exists. Note that if VEDBOPEN is being called from a COMMAND file, MAIN.PUB.VESOF should be specified in a \$PROGRAM clause. However, if VEDBOPEN is being called from an EXPRESSION PROGRAM, the name of the EXPRESSION PROGRAM should be specified in a \$PROGRAM clause.

Determining the status of a VEDBxxx call

All functions (except for VEDBOPEN and VEDBGET) return TRUE if all went well, FALSE if there was an error (EOF, BOF, end of chain, beginning of chain, no such key, duplicate key, etc.) and abort in case of some other error. The predefined variable VEDBERR will be set to word 0 of the status array after each call to a VEDBxxx function and can be tested for the exact error. We also display the DBERROR message in case of an error.

In addition to VEDBERR, the following predefined variables can be used (where appropriate, depending upon which VEDBxxx function was called) to retrieve information that is returned in the STATUS array.

Variable	Status Word
VEDBDATALEN	1
VEDBRECNUM	1
VEDBCHAINLEN	2
VEDBPREVREC	3
VEDBNEXTREC	4

Please consult your HP TurboIMAGE Reference Manual for details about each of these status array elements as well as the corresponding IMAGE intrinsics for the following VEDBxxx functions.

Function	Type	IMAGE Call
VEDBOPEN (<i>'dbname, pass, mode'</i>)		
	[STR]	DBOPEN. Accepts a string containing the database name, password and mode (delimited by commas) and returns a string value that can then be used as the <i>dbid</i> in the other VEDBxxx functions. Remember to close the base using VEDBCLOSE.
VEDBFIND (<i>dbid, dset, [mode,] item, key</i>)		
	[BOOL]	DBFIND. Sets up pointers to the first and last entries of a detail chain to prepare for subsequent VEDBGETs. The <i>dset</i> parameter must be the name of a DETAIL dataset. The item is the name of the desired search item. The key parameter is a string containing the value of the search item in the desired chain. The optional <i>mode</i> parameter is used to access the TPI (Third Party Indexing) feature provided by TurboIMAGE/iX starting with MPE/iX 4.0.
VEDBGET (<i>dbid, dset, mode, list[, key]</i>)		
	[STR]	DBGET. Returns a string containing the items requested in the list parameter. Mode is an integer that determines the reading method (see your HP TurboIMAGE Reference Manual for details). Optionally, a key value (which must be the correct length) may be specified to support modes 4, 7 and 8. If you are using mode 4 and want record 100, use STRWRITE(100: 'I4') as the key value. This function sets the special variable VEEOF to TRUE if the desired entry couldn't be found or EOF is reached. You can retrieve a maximum of 255 characters with a single VEDBGET.
VEDBGETV (<i>'varname', dbid, dset, mode, list[, key]</i>)		
	[BOOL]	DBGET. Same as VEDBGET except that it returns a BOOLEAN value (just like the other functions). The requested item values are assigned to the variable specified in the first parameter. (Also sets VEEOF TRUE if entry couldn't be found or EOF is reached).

Function	Type	IMAGE Call
Note: IMAGE requires you to pass it a key of the correct length, space-filled; use <code>STRWRITE(<i>str:len</i>)</code> if necessary.		
VEDBPUT (<i>dbid, dset, list, buffer</i>)		
	[BOOL]	DBPUT. Adds a new entry to a dataset. If the database was opened with VEDBOPEN, it must have been opened in access mode 1, 3 or 4.
VEDBPUTL (<i>dbid, dset, list, buffer</i>)		
	[BOOL]	DBPUT, auto-lock. Same as VEDBPUT except that it automatically does a DBLOCK and DBUNLOCK around the DBPUT. This makes it more convenient to use the auto-open form of <i>dbid</i> .
VEDBUPDATE (<i>dbid, dset, list, buffer</i>)		
	[BOOL]	DBUPDATE. Modifies the values of the data items specified in list in the current record. The database must be open in access mode 1, 2, 3 or 4.
VEDBUPDATEK (<i>dbid, dset, list, buffer, key</i>)		
	[BOOL]	DBLOCK, DBGET, DBUPDATE, DBUNLOCK. Allows quick updating of master datasets by specifying the a string containing the key value. It automatically does a DBLOCK, DBGET and DBUNLOCK for you. This makes it more convenient to use the auto-open form of <i>dbid</i> .
VEDBDELETE (<i>dbid, dset</i>)		
	[BOOL]	DBDELETE. Deletes the current entry from a dataset. The database must be open in access mode 1, 3 or 4.
VEDBDELETEK (<i>dbid, dset, key</i>)		
	[BOOL]	DBLOCK, DBGET, DBDELETE, DBUNLOCK. Same as VEDBDELETE but allows the quick deletion of master dataset entries by specifying a string containing the key value. It automatically does a DBLOCK, DBGET and DBUNLOCK for you. This makes it more convenient to use the auto-open form of <i>dbid</i> .
VEDBLOCK (<i>dbid, dset, mode</i>)		
	[BOOL]	DBLOCK. Modes 1-4 are supported. Be careful using mode 1 or mode 3; they could cause a deadlock.

Function	Type	IMAGE Call
VEDBUNLOCK (<i>dbid</i>)		
	[BOOL]	DBUNLOCK. Relinquishes the locks acquired by all previous calls to VEDBLOCK.
VEDBCONTROL (<i>dbid, mode</i>)		
	[BOOL]	DBCNTROL. Allows a process accessing the database in exclusive mode (VEDBOPEN mode 3) to enable or disable the 'output deferred' option. The mode parameter is an integer that can be either 1 or 2. 1 enables the output deferred option, 2 disables it. See HP's TurboIMAGE Reference Manual for details.
VEDBCLOSE (<i>dbid, dset, mode</i>)		
	[BOOL]	DBCLOSE. Terminates access to a database or a dataset or re-initializes access to a dataset (mode 1, 2 or 3 respectively). To close the database, specify an empty string (i.e. "") for the <i>dset</i> parameter. See HP's TurboIMAGE Reference Manual for details.

Examples: To assign the EMP-NAME field of the entry in dataset F00 of database MYDB whose key is EMPID to the variable EMP_NAME:

```
SETVAR EMP_NAME VEDBGET ('MYDB,,5', 'F00', 7, 'EMP-NAME', EMPID)
```

Here is an MPEX command file that will accept a key value as a parameter and then traverses the detail chain; printing out some of the fields in each detail entry:

```
PARM CODE
SETVAR DBID VEDBOPEN('CUST.VESOFT.LIST,VESOFT,5')
IF NOT VEDBFIND(DBID,'TECH','CODE',UPS("!CODE")) THEN
    ECHO CANNOT FIND ANY DETAIL ENTRIES FOR CODE: !CODE
ENDIF
WHILE VEDBGETV('REC',DBID,'TECH',5,"OPEN'DATE,QUEST'ABOUT")
    ECHO DATE: ![CDATE(DATEPARSE(REC[0:8],"YMD"),"MM/DD/YY")]&
    ABOUT: ![REC[10:50]]
ENDWHILE
IF NOT VEDBCLOSE(DBID,'TECH',1) THEN
    ECHO UNABLE TO CLOSE THE BASE!
ENDIF
DELETEVAR DBID,REC
```

Appendix B

Here is an example of a STREAMX prompt that checks if the entered code exists as a KEY value in a master dataset:

```
::PROMPT STRING KEYVAL="Customer code"&  
:: ;CHECK=VEDBGETV('DUMMYVAR', 'CUST.VESOFT.LIST,VESOFT,5', &  
:: 'CUST', 7, ' ', STRWRITE(UPS(KEYVAL):6)) &  
:: ;CHECKERR= 'Invalid customer code.'
```

Trapping program output (\$STDLIST)

Say that you want to run a program (or do an MPE command) that generates a lot of output, but only want to see some of the output (e.g. a particular line or only the error messages) or want to preprocess the output before seeing it (e.g. extract only a particular column range).

You could, of course, execute the program or command with \$STDLIST redirected to a disc file (issuing the proper :FILE equations, if needed), do a VEFREAD of the file or a %PRINT ;SEARCH=, ;CONTEXT=, or ;FORMAT= and then purge the file.

However, this (including the inevitable fumbling with :FILE equation parameters) is often complicated enough that people just don't bother doing it.

With MPEX, this can be done very easily, e.g.

```
%PRINT (![STDLIST('RUN MYPROG')]);SEARCH="ERROR";CONTEXT=,2;&  
FORMAT=STRWRITE(R:'GARBAGE')
```

This does a RUN MYPROG with STDLIST= redirected to a special (uniquely named) temporary file, prints only those lines in that file that contain "ERROR" (;SEARCH="ERROR"), plus two lines after each (;CONTEXT=,2), with unprintable characters replaced by dots (;FORMAT=STRWRITE(R:'GARBAGE')) and automatically purges the temporary file.

The key to all this is the STDLIST(*S*) function, which executes the command contained in the string *S* with \$STDLIST redirected to a temporary file and returns a string containing all the FOPEN parameters needed to read this file and delete it when it's done. This string (which is essentially a :FILE equation without the word "FILE") can then be passed to a VEFxxx function, or to a %PRINT(xxx) command (the parentheses around the ![STDLIST(...)] are necessary; otherwise %PRINT would expect only a filename, not all the FOPEN parameters). For instance, you might say:

```
%SETVAR S VEFREAD(STDLIST('SHOWTIME'))
```

which will set *S* to be the first line of a :SHOWTIME listing.

By default, the list file is built as ;REC=-508,,V,ASCII;NOCCTL;DISC=15990,32. If you want to change one of these parameters — for example, make the file fixed-record length, so that you can then do a VEFREADDIR against it — you can specify this as a second parameter to STDLIST, e.g.

```
%SETVAR REC VEFREADDIR(STDLIST('RUN MYPROG','REC=-80,,F,ASCII'),7)
```

Of course, if you need to get more than one record from the file, you can VEFOPEN the file and do VEFREADs or VEFREADDIRs using the file number. Remember, though — when you do the VEFCLOSE, the file will get automatically purged.

Note: The STDLIST(*S*) function is implemented via the ">" I/O re-direction mechanism (see ["Redirecting command input and output"](#) in the MPEX User Manual). Since I/O redirection is not allowed in the CALC (and other commands), you can't use the CALC command in the STDLIST(*S*) function.

Terminal input

Sometimes — usually from within command files or flexible LISTF template files — you might want to prompt the user for input. There are several functions that do this: (*F* is a special "format string" that is described later — it can be omitted unless otherwise stated below and defaults to the empty string):

Function	Type	Result
READSTRING (<i>S</i> [, <i>F</i> [, <i>I</i>]]) [STR]		Outputs the prompt message <i>S</i> and inputs a string (which it returns). <i>I</i> is the maximum input length (default 256) (<i>F</i> is required if <i>I</i> is specified, but can be just an empty string, i.e. "").
READINTEGER (<i>S</i> [, <i>F</i>]) [INT]		Outputs the prompt message <i>S</i> and inputs an integer (which it returns).
READDATE (<i>S</i> [, <i>F</i>]) [DATE]		Outputs the prompt message <i>S</i> and inputs a date (which it returns).
READTIME (<i>S</i> [, <i>F</i>]) [TIME]		Outputs the prompt message <i>S</i> and inputs a time (which it returns).
READREAL (<i>S</i> [, <i>F</i>]) [REAL]		Outputs the prompt message <i>S</i> and inputs a real number (which it returns).
READSELECT (<i>S</i> , <i>T</i> [, <i>F</i>]) [STR]		Outputs the prompt message <i>S</i> and inputs a single character, which must be one of the characters in string <i>T</i> ; returns the character input. If carriage return was hit, returns the first character of <i>T</i> (unless the NOCR flag was specified in <i>F</i>). <i>T</i> should not be a null string ("").

If, for example, you use the expression

```
READINTEGER("ENTER A NUMBER? ")
```

Appendix B

you will be prompted for input until you enter a valid number:

```
ENTER A NUMBER? XYZZY
An integer value between -2147483648 and 2147483647 expected. (CIERR 10010)
ENTER A NUMBER? 1.2
An integer value between -2147483648 and 2147483647 expected. (CIERR 10010)
ENTER A NUMBER? 55
```

In this case, the function's result will be 55 (the number input).

The "format string" indicated as *F* may contain one or more format specifiers, delimited by commas. Valid format specifiers are:

Format Specifier	Result
NOECHO	Don't echo the input.
NOCR	Force the user to input something, not just hit CR.
RTRIM	String input: strip trailing blanks from input data.
UPSHIFT	String input: upshift input data.
DOWNSHIFT	String input: downshift input data.
HEX	Integer input: assume input is hexadecimal.
OCTAL	Integer input: assume input is octal.
YMD	Date input: assume input is [YY]YY/MM/DD (default).
MDY	Date input: assume input is MM/DD/[YY]YY.
DMY	Date input: assume input is DD/MM/[YY]YY.

Thus, saying

```
READINTEGER ("SECRET NUMBER? ", "HEX,NOCR,NOECHO")
```

would prompt for an integer without echoing (NOECHO), assuming the input is a hexadecimal number (HEX) and forbidding the user to just hit a CR (NOCR).

If the format string parameter is omitted, none of the options above will be enabled (except for date input, for which YY/MM/DD input will be selected).

Terminal input: VEMODIFY

In addition to prompting for strings, integers and other data, we provide an expression function that allows you to modify strings using the same EDITOR/MPE modification commands or our Visual Modify keystrokes. The `VEMODIFY(...)` function takes a string as a parameter, displays the string and allows the user to modify it, then returns the modified string to your expression program or string variable. By default, normal MPE/EDITOR modification commands are accepted ([I]nsert, [D]elete or [R]eplace characters), but if you have the `HPREDOVEMODIFY JCW` set to 1, our standard Visual Modify will be used. Refer to MPEX help on `%REDO` for a description of the keystrokes available in "visual modify" mode.

Output formatting

One function in particular (called `STRWRITE`) is a very powerful tool for formatting a number of objects (strings, integers, dates, etc.) in a number of different ways (left-justified, right-justified, decimal, hexadecimal, etc.).

The general format of `STRWRITE` is:

```
STRWRITE (descriptor, descriptor, ..., descriptor)
```

where each descriptor is one of

```
expression                or
expression: fieldlen      or
expression: formatstring  or
expression: fieldlen: formatstring
```

`STRWRITE` returns a string which contains the formatted values of all the specified expressions.

This is more easily shown than explained:

```
STRWRITE ('DECIMAL', 123:6, ' IS OCTAL ', 123:'OCTAL',
          ' AND HEXADECIMAL ', 123:5:'HEX')
```

This operation returns the string

```
'DECIMAL   123 IS OCTAL 173 AND HEXADECIMAL    7B'
```

which is a concatenation of

```
'DECIMAL'+
'   123'+ (123 formatted in a field of length 6)
' IS OCTAL '+
'173'+   (123 formatted with format descriptor 'OCTAL')
' AND HEX '+
'   7B'  (123 formatted in a field of length 5 with format descriptor 'HEX')
```

An expression may be specified without any qualifiers (e.g. `'DECIMAL'`), in which case it's printed in some default format in a field of minimum length; it may be specified with an integer field length, in which case it's printed (left- or right-justified depending on the type) in a field of the given length; it may be specified with a string format specifier, in which case it's printed with appropriate formatting in a field of minimum length; or, it may be specified with both.

String formatting

When you do a `STRWRITE` of a string, you have several options:

<code>STRWRITE (S)</code>	Return the string <i>S</i> verbatim.
<code>STRWRITE (S: I)</code>	Place the string <i>S</i> left-justified in a field of <i>I</i> characters, e.g. <code>STRWRITE("FOO":5)</code> is <code>"FOO "</code> .

<code>STRWRITE (S: I: 'RIGHTJUST')</code>	Place the string <i>S</i> right-justified in a field of <i>I</i> characters, e.g. <code>STRWRITE("FOO":5:"RIGHTJUST")</code> is " FOO".
<code>STRWRITE (S: I: 'CENTER')</code>	Center the string <i>S</i> in a field of <i>I</i> characters, e.g. <code>STRWRITE("FOO":5:"CENTER")</code> is " FOO ".
<code>STRWRITE (S: 'UPSHIFT')</code>	Return the string <i>S</i> upshifted.
<code>STRWRITE (S: 'DOWNSHIFT')</code>	Return the string <i>S</i> downshifted.
<code>STRWRITE (S: 'GARBAGE')</code>	Return the string with garbage (non-printable) characters replaced by ".s".

Any of the above options may be combined, e.g.

```
STRWRITE ( 'FOO':10:'CENTER,UPSHIFT,GARBAGE' ) .
```

Integer formatting

When you do a `STRWRITE` of an integer, you have several options:

<code>STRWRITE (I)</code>	Return the string representation of the integer <i>I</i> , e.g. <code>STRWRITE(10)</code> is "10".
<code>STRWRITE (I1: I2)</code>	Return the string representation of <i>I1</i> right-justified in <i>I2</i> -character field, e.g. <code>STRWRITE(10:5)</code> is " 10".
<code>STRWRITE (I1: I2: 'LEFTJUST')</code>	Return the string representation of <i>I1</i> left-justified in <i>I2</i> -character field, e.g. <code>STRWRITE(10:5:"LEFTJUST")</code> is "10 ".
<code>STRWRITE (I: 'OCTAL')</code>	Returns the octal representation of <i>I</i> , e.g. <code>STRWRITE(123:"OCTAL")</code> is "173".
<code>STRWRITE (I: 'HEX')</code>	Returns the hexadecimal representation of <i>I</i> , e.g. <code>STRWRITE(123:"HEX")</code> is "7B".
<code>STRWRITE (I1: I2: 'ZEROFILL')</code>	Returns <i>I1</i> right-justified in <i>I2</i> characters with leading zeros, e.g. <code>STRWRITE(123:5:"ZEROFILL")</code> is "00123".
<code>STRWRITE (I: 'UNSIGNED')</code>	Returns <i>I</i> formatted as an unsigned integer, e.g. <code>STRWRITE(-1:"UNSIGNED,HEX")</code> is "FFFFFFFF".
<code>STRWRITE (I: 'I2')</code>	Returns the 2-byte representation of the low-order 16-bits of the integer <i>I</i> . Useful for putting integer values into records to be written to files using the <code>VEFWRITExxx</code> functions.
<code>STRWRITE (I: 'I4')</code>	Returns the 4-byte representation of the 32-bit integer <i>I</i> . Useful for putting integer values into records to be written to files using the <code>VEFWRITExxx</code> functions.

Again, any of the above options may be combined, e.g.

```
STRWRITE(I:10:"LEFTJUST,HEX,UNSIGNED").
```

Date formatting

The format specifiers used for formatting dates and times are quite different from those used for formatting strings and integers. Here's an example of a `STRWRITE` of a date:

```
STRWRITE(TODAY: '%1W, %1M %RD, 19%0Y')
```

On 2/29/68, this would have returned

```
THU, FEB 29, 1968
```

Why? Well, the commas, the spaces and the "19" are transferred verbatim to the generated string. However, certain special specifiers (largely those start with "%") are replaced by one of several things: (Examples below assume the date Friday, 2 January, 1903)

Format Specifier	Example	Result
%0D	"02"	The day of month, zero-filled, 2 characters.
%0M	"01"	The month number, zero-filled, 2 characters.
%0Y	"03"	The year, zero-filled, 2 characters.
%0C	"19"	The 4-digit year's high-order 2-digits.
%4Y	"1903"	The 4-digit year.
%RD, %RM, %RY	" 2"	Same as %0D/%0M/%0Y, but right-justified (i.e. padded with blanks).
%D, %M, %Y	"2"	Same as %0D/%0M/%0Y, but not padded either with blanks or with zeros.
%1M	"JAN"	The month, 3-letter abbreviation.
%2M	"January"	The month's full name (first char UPPERCASE).
%1W	"FRI"	The day-of-week, 3-letter abbreviation.
%2W	"Friday"	The day-of-week's full name (first char UPPERCASE).
DD	"02"	Same as %0D.
MM	"01"	Same as %0M.
YY	"03"	Same as %0Y.
CC	"19"	Same as %0C.

All format specifiers must be UPPERCASE; for instance, "%1w" is invalid — you must use "%1W".

Appendix B

Some useful date formats might be:

```
STRWRITE (D: 'YY/MM/DD')           03/01/02
STRWRITE (D: 'MM/DD/YY')           01/02/03
STRWRITE (D: 'DD.MM.YY')           02.01.03
STRWRITE (D: '%1W, %1M %RD, 19%0Y') FRI, JAN  2, 1903
STRWRITE (D: '%2W, %2M %0D, %0C%0Y') Friday, January 02, 1903
STRWRITE (D: '%4Y:MM:DD')           1903:01:02
```

Time formatting

Time format specifiers are quite similar to the date format specifiers. Special format specifiers indicate parts of the time that are to be included into the formatted string. For instance,

```
STRWRITE(RSTTIME: '%rh:%0m:%0s %x')
```

would, if RSTTIME were 7:33:05 PM, return

```
7:33:05 PM
```

More precisely, the valid specifiers are: (examples are based on a time of 9:07:00 PM)

Format Specifier	Example	Result
%0h	"09"	The hour (by 12-hour clock), zero-filled, 2 chars.
%024h	"21"	The hour (24-hour clock), zero-filled, 2 chars.
%0m	"07"	The minute, zero-filled, 2 chars.
%0s	"00"	The second, zero-filled, 2 chars.
%rh, %r24h, %rm, %rs	" 9"	Same as the "%0..." format, but right-justified in 2 characters (i.e. padded with spaces)
%h, %24h, %m, %s	"9"	Same as the "%0..." format, but without leading spaces or zeros.
%x	"PM"	Either "AM" or "PM" depending on the hour.

All format specifiers must be lowercase; for instance, "%H" is invalid — you must use "%h".

Real number formatting

Real number formatting is a bit different from all the other formatting constructs. Instead of being able to specify the field length and/or a format specifier string, you may specify a field length and optionally another integer indicating the number of digits to be printed after the decimal point. Thus,

```
STRWRITE(1.2)           yields " 1.20"
STRWRITE(1.2:11)        yields " 1.20"
STRWRITE(1.2:11:4)      yields " 1.2000"
```

As you see, the number after the first ":" is the field length (default 9) and the number after the second ":" is the number of digits after the decimal point (default 2).

Advanced expression programming

The expression operators and functions that we described above had to do with what we might call "simple expressions" — expressions that operate on several things (constants, variables, etc.) and return a value. MPEX's %CALC, %SETVAR, %IF, STREAMX's CHECK=... conditions, etc. all use "simple expressions" like that.

In certain circumstances, though — for instance, in the flexible LISTF files — you want more than just an expression. What you want is more like a program, with a sequence of statements, each to be executed in turn. This program should be able to have variables, assignment statements, IF statements, WHILE loops, WRITELN statements to output things, etc. — all the power of a true programming language.

Here, for instance, is a snippet from what might be an MPEX flexible LISTF file:

```
VAR I: INTEGER;
VAR S: STRING[120];

I:=10;
...
IF I>20 AND S<>'FOO' THEN
  WHILE I<>0 DO
    BEGIN
      WRITELN (S);
      ...
    END
  ELSE
    ...;
```

As you see, this looks much like a PASCAL program and has most of the power of PASCAL. Statements are separated by ";"s; variables can be declared, assigned to and used; IF... THEN... ELSE and WHILE... DO loops are available; BEGIN... END can be used to group statements.

In addition to flexible LISTF files, MPEX expression programs can be used to do advanced tasks that regular command files are too slow for (although expression programs are still slower than regular, compiled programs). For more details on writing expression programs, see ["Advanced expression programming"](#) below.

IF ... THEN ... ELSE

Conditional processing is quite simple; in fact, it's almost identical to PASCAL's:

```
IF condition THEN
  statement
ELSE
  statement;
```

Appendix B

For instance,

```
IF I>20 AND S MATCHES 'X@' THEN
  WRITELN ('HELLO')
ELSE
  WRITELN ('GOOD BYE');
```

To use more than one statement in either the THEN block or the ELSE block, use BEGIN...END:

```
IF I>20 AND S MATCHES 'X@' THEN
  BEGIN
    WRITELN ('HELLO');
    WRITELN ('THERE')
  END
ELSE
  BEGIN
    WRITELN ('GOOD');
    WRITELN ('BYE')
  END;
END;
```

One new feature of the IF... THEN... ELSE is the ability to use them in an "expression" context, i.e. to return a value:

```
WRITELN (IF I>20 THEN 55 ELSE 66);
```

Here, either 55 or 66 will be output, depending on whether I>20 is true or false.

If you want an IF to return a value, the THEN block and the ELSE block must return values of the same type —

```
WRITELN (IF I>20 THEN 55 ELSE 'FOOBAR');
```

is illegal, since we have to know the type of the expression to be WRITELNed at compile time.

Of course, if the IF doesn't return a value, it doesn't matter what the types of the THEN and ELSE blocks are (in fact, they can then be BEGIN... END constructs, which have no type).

WHILE ... DO

Our WHILE...DO loops are identical to PASCAL's. If you say

```
WHILE condition DO
  statement;
```

then the statement will be executed again and again as long as the condition is true. BEGINs and ENDs can be used if you want to execute more than one statement:

```
WHILE I<=100 DO
  BEGIN
    WRITELN (I:10, I*I:10, I*I*I:10);
    I:=I+1;
  END;
```

Constructs like REPEAT...UNTIL or a FOR *x* := *y* TO *z* DO can be easily emulated with the WHILE loop.

FOR *stringvar* IN FILE (*fopenparms*) DO

This form of our FOR loop allows you to process all of the records in a file. The full syntax is:

```
FOR stringvar IN FILE (fopenparms) DO
    statement;
```

For example, to search through a list of files and report on any files that are missing:

```
VAR FILENAME_TO_TEST: STRING[68];
FOR FILENAME_TO_TEST IN FILE ("FILELIST,OLD") DO
    IF NOT FEXISTS(FILENAME_TO_TEST) THEN
        WRITELN ("File ",RTRIM(FILENAME_TO_TEST)," is missing!");
```

Here is a more advanced example that implements a new command. If you keep this as a file named PURGETMP (with filecode 171 — see [Writing expression programs](#)), you can say

```
%PURGETMP @.@.@
```

to purge all of your temporary files.

Here is the expression program itself:

```
VAR TEMPFILENAME: STRING[68];
FOR TEMPFILENAME IN FILE (STDLIST('LISTFTEMP '+VEPARMS)) DO
    BEGIN
        IF TEMPFILENAME MATCHES "@.@.@" THEN BEGIN
            WRITELN ("-----PURGETMPing ",RTRIM(TEMPFILENAME));
            MPE ("PURGE "+TEMPFILENAME+",TEMP");
        END;
    END
```

What does all of this mean?

- Well, VEPARMS is a predefined variable containing the parameters passed to the expression program;
- the STDLIST function (documented earlier in this Appendix) executes an MPE command (in this case, "LISTFTEMP"), writes the output of the command to a temporary file and returns the fopenparms needed to open that file and then automatically purge it when you close it;
- "FOR TEMPFILENAME IN FILE..." reads the file created by the STDLIST function and puts each line in turn into the variable "TEMPFILENAME";
- we check to make sure TEMPFILENAME contains a filename (to skip past the heading portion of the output of the LISTFTEMP command) and if it does,
- we show the user the name of the file ("WRITELN..."), and, finally,
- we call the MPE function to purge the file!

TRY ... RECOVER and TRY ... CLEANUP

If you want to do your own error handling in the file, you can use two constructs — TRY...RECOVER (analogous to the %TRAPERROR/%IFERROR) and TRY...CLEANUP (analogous to the %TRAPERROR/%CLEANUP).

```
TRY
    stmt1
RECOVER
    stmt2
```

does *stmt1* (which can be either a single statement or a BEGIN/END block) and if an error occurs, does *stmt2* and then continues with the remainder of the program.

```
TRY
    stmt1
CLEANUP
    stmt2
```

does *stmt2* after doing *stmt1*, whether or not an error occurred in *stmt1*; however, if an error did occur, it aborts the program after doing *stmt2*. Thus, a common sequence that you might use for file handling would be:

```
FNUM:=VEFOPEN ...;
TRY
    BEGIN
        ...
    END
CLEANUP
    VEFCLOSE (FNUM);
```

Expression program variables

We support variables of several types:

INTEGER	32-bit integer.
BOOLEAN	TRUE or FALSE.
REAL	32-bit floating point (single precision real).
DATE	Date.
TIME	Time.
STRING[<i>xxx</i>]	Variable length string of up to <i>xxx</i> characters.

To declare a variable, just say

```
VAR varname: type;
```

e.g.

```
VAR LINECOUNT: INTEGER;
VAR HEADER: STRING[256];
```

You must have exactly one variable per VAR statement — both

```
VAR LINECOUNT, XYZZY: INTEGER;          (* illegal! *)
```

and

```
VAR LINECOUNT: INTEGER; XYZZY: INTEGER;  (* illegal! (no second VAR) *)
```

are illegal. Instead, say

```
VAR LINECOUNT: INTEGER;
VAR XYZZY: INTEGER;
```

All VAR statements must come at the **beginning** of the "expression program" in which they'll be used.

Once you've declared a variable, you can assign to it by saying

```
variable := expression;
```

e.g.

```
I := I * 17;
S := 'TESTING ONE TWO THREE';
```

The left-hand side of an assignment statement can only be

- a variable (`I := 17`) or
- a substring (`S[3:4] := 'XYZZ'`) — in this case, the entire substring must be within the bounds of the string.

In particular, neither expressions nor MPEX variables (or MPE/iX variables) may be put on the left-hand side of an assignment statement. To set MPEX variables, use the `BVARSET`, `IVARSET` and `SVARSET` functions or the MPEX `%SETVAR` command.

After you've assigned a value to a variable, you can use it much as you'd be able to use a file attribute variable or an MPEX variable:

```
IF I>10 THEN
  WRITELN (S);
```

Appendix B

Note that variables declared with a VAR command are strictly local variables, local to the "expression program" that declares them. They can not be later referenced in an MPEX command, whether in a %CALC, %SETVAR, ! substitution construct or what have you.

Sorry, no record structures (yet?). Also, be wary of declaring long strings! You have a very limited amount of space left (if you exhaust it, it's STACK OVERFLOW time).

Outputting data

Every good language should have a good input and output mechanism (SPL is a notable offender in this area). For input, you can use the READ_{xxx} functions discussed [above](#) (READINTEGER, READSTRING, READDATE, etc.). For output, you can use:

```
WRITELN (value1, value2, ..., valueN);
```

and

```
WRITEPROMPT (value1, value2, ..., valueN);
```

Both are just like STRWRITE except that they output to the terminal (or whatever the list file of the expression program is) instead of returning a string. WRITELN outputs the data and then a carriage return/line feed; WRITEPROMPT outputs it without a CR/LF.

```
WRITE (value1, value2, ..., valueN);
```

formats all the values but doesn't actually output them. Instead, the formatted text is kept in the "output buffer"; subsequent WRITES, WRITELNs and PROMPTs will append to this output buffer, until it's actually output (and cleared) by a WRITELN or a WRITEPROMPT.

Thus,

```
WRITE ('FOO');  
WRITE ('BAR');  
WRITELN ('XYZ');
```

will output "FOOBARXYZ".

If you have stuff in the output buffer and you want to throw it away, you can say

```
CANCELWRITE ();
```

(the open and close parentheses are important).

Other I/O constructs include:

```
* WRITEPAGE ()
```

This causes a form feed on the expression program output file.

Comments

You can insert comments in expression programs by enclosing them in "(*" and "*)". For example,

```
...
(* Figure out the month number for a three-letter abbreviation *)
MONTH:=(POS(UPS(MONTHSTR[0:3]),
    "JAN FEB MAR APR MAY JUN JUL AUG SEP OCT NOV DEC ")
    +3)/4;  (* Remember, POS starts counting at 1 *)
...
```

This feature may also be used to add comments to the following files:

```
SECURCON.DATA.VESOFT
STREAMX.DATA.VESOFT
LOGOFF.DATA.VESOFT
LOGOFFAB.PUB.VESOFT
LOGOFFWR.PUB.VESOFT
```

Writing expression programs

Especially now that MPEX can access files, there are many things that you can do in MPEX that you used to need a program to do. However, MPEX command files are much slower than programs, because every command has to be interpreted; thus, using a %WHILE loop to read a file might take you a tenth of a second or more per record.

You can do things much quicker (though still more slowly, perhaps by as much as a factor of 10 or more, than in a program) using VESOFT expression programs. You may have already used expression programs if you've written flexible LISTF files — they are much like simple VESOFT expressions, in that you can use all the functions that you can use in %CALC, %IF, etc., but you can also declare variables and use constructs like IF/THEN/ELSE, WHILE, BEGIN/END, etc. — these are all documented above under "Advanced Expression Programming".

In order to create an expression program, simply write it with your favorite text editor and /KEEP it as a file with filecode 171 (you can use MPEX's %ALTFILE *files*et;CODE=171). You can then execute it by just typing its name at the % prompt, just like a command file (HPPATH is honored for this just as it is for command files and programs).

The first time you execute any expression program, MPEX "compiles" it into a file with the same name, in the same group and account, but with a "9" appended to the end of the filename (or the last character changed to a "9" if the filename is already 8 characters long; or, if the filename is 8 characters and already ends in 9, the new file will end in "8"). This new file will have filecode 176.

If you are using the Native Mode version of MPEX, compiled expression programs will end in 9 (or 7) instead of 9 (or 8).

Note: You must be careful not to create two different expression programs in the same *group.account* with the filename the same for the first 7 characters.

Appendix B

For safety's sake, if there is an existing file with the filename that the expression program is supposed to be compiled into, but it doesn't have filecode 176, MPEX will not overwrite, but will instead print an error message (your new expression program will not run). If this happens, simply PURGE or RENAME the "9"-file.

For example, if you create an expression program named MYEXPROG.MYGROUP.MYACCT (with file code 171), then type

```
%MYEXPROG
```

MPEX will compile MYEXPROG into a new file named MYEXPRO9.MYGROUP.MYACCT (for CM users) or MYEXPRO8.MYGROUP.MYACCT (for NM users), with filecode 176 and then execute it. The next time you execute it (provided you haven't modified MYEXPROG in the meantime), it will execute much faster, since it has already been "compiled". Note that you never say %MYEXPRO9 or %MYEXPRO8; always use the name of the original file (in this case, MYEXPROG).

Whenever you make a change to any expression program, it will be automatically re-compiled the next time you execute it. This "pre-compiling" helps make expression programs execute much faster.

Any parameters that were specified are passed to the command file as the variable VEPARMS (it is your job to parse it, probably using TOKEN and REMTOKEN). Here's an example:

```
VAR S: STRING[256];
VAR I: INTEGER;
VAR FNUM: INTEGER;

I:=0;
FNUM:=VEFOPEN (VEPARMS+',OLD');
S:=VEFREAD (FNUM);
WHILE NOT VEEOF DO
  BEGIN
    I:=I+INTEGERPARSE (S);
    S:=VEFREAD (FNUM);
  END;
VEFCLOSE (FNUM);
WRITELN ('Sum of lines is ', I);
```

This expression program opens a file (whose name is given as the parameter to the expression program) and sums the integer values stored in all of its records — just an example of what you can do. An alternative to the above would be:

```
VAR S: STRING[256];
VAR I: INTEGER;
I:=0;
FOR S IN FILE (VEPARMS+',OLD') DO
  I:=I+INTEGERPARSE (S);
WRITELN ('Sum of lines is ', I);
```

The FOR *stringvar* IN FILE (*fopenparms*) DO *statement* performs *statement* for each record in the file, assigning the value of the record to *stringvar*. This saves you from

having to do your own VEFOPEN, VEFREAD and VEFCLOSE and also properly closes the file in case an error occurs.

A very important disclaimer

I hesitated a lot before releasing this feature, since it should not be viewed as a general-purpose programming language. It is limited in a number of ways and you should be aware of these ways before you invest any serious time into using it.

- It is and always will be, slower — maybe much slower — than a third-generation-language program. Though we "compile" the expression program, we compile it into a kind of pseudo-code that we then interpret.
- Expression programs lack many features that are necessary for a really general-purpose programming language, e.g. record structures, procedures, etc. On the other hand, it does have a lot of nice things — like convenient file access and string manipulation — that most languages don't have.

Although we'll entertain enhancement requests to work around these problems, they'll have a substantially lower priority than other enhancement requests and we'll often expect you to live with workarounds that might be less than perfect. Historically, expression programs were developed to allow VESOFT to implement its own features (like most MPEX commands, all the %SEC commands and a very great part of VEAUDIT) and we've made them available to users because we had already created them for ourselves. Especially because we expect only rather sophisticated users to use expression programs, we probably won't implement many features unless they're useful to ourselves.

One other note — curious people may look at files in the VESOFT account and try to use in their own expression programs the features that they see us use in ours. However, for security reasons, the features we make available to users are a subset of those features we use internally. Some of those features will actually give you errors if you try to use them yourselves; even if, by accident, you can get them to work, we will not support them unless they are documented.

We've put in these disclaimers because we don't want to create any expectations that we can't live up to or have you invest a lot of effort into trying to do things with this feature that it was never intended to do. On the other hand, for short, simple problems, we feel that expression programs can be a really powerful solution.

Appendix C: LOADER ERROR MESSAGES EXPLAINED

Loader error messages, usually generated at `:RUN` command execution time, can best be described in one word: CRYPTIC. They usually do not tell you what the problem is and they never tell you how to remove or get around the problem. The Error Messages and Recovery Manual or the Error Messages appendix of the System Intrinsic Manual is not much help either — more often than not, no explanation is given, save a repetition of the text of the message.

The following is a feeble effort at cataloging explanations and workarounds to some of the more frequently occurring error messages:

20 ILLEGAL LIBRARY SEARCH (LOAD ERR 20):

Either 1) An attempt was made to run a program in the PUB group of the SYS account with LIB=G or LIB=P or 2) an invalid library search parameter was specified in the CREATE or LOADPROC intrinsic call.

WORKAROUND: In case 1) since LIB=G, LIB=P and LIB=S (default) are functionally equivalent for programs in PUB.SYS, the program should be run with LIB=S or the LIB= clause should be omitted; in case 2, a valid library search parameter should be specified.

21 UNKNOWN ENTRY POINT (LOAD ERR 21):

A program was run with an entry point which does not exist in the program file.

WORKAROUND: Specify an entry point that exists in the file.

22 TRACE SUBSYSTEM NOT PRESENT (LOAD ERR 22):

Your SPL or FORTRAN program uses the \$TRACE keyword, but TRACE/3000 is not installed on your system.

WORKAROUND: Remove the \$TRACE keyword and recompile. You don't REALLY use TRACE/3000, do you?

23 STACK SIZE TOO SMALL (LOAD ERR 23):

The stack size with which the program was to be run was less than 512.

WORKAROUND: Specify a bigger stack size.

24 MAXDATA TOO LARGE (LOAD ERR 24)

This error is no longer used.

25 STACK SPACE REQUIRED EXCEEDS SPECIFIED MAXDATA (LOAD ERR 25):

The maxdata specified is not large enough to fit the initial stack of the program to be run.

WORKAROUND: Specify a larger maxdata or decrease the program's stack usage.

26 PROGRAM LOADED IN OPPOSITE MODE (LOAD ERR 26):

An attempt was made to run with the `;NOPRIV` keyword a program that is already loaded (run or allocated) without the `;NOPRIV` keyword, or vice versa.

WORKAROUND: Run the program with or without the `;NOPRIV` keyword, whichever is appropriate.

27 SL BINDING ERROR (LOAD ERR 27):

The program to be loaded has external references to procedures that cannot be found in the SL files with which the program is to be loaded.

WORKAROUND: Correct the program so it will not refer to those procedures, add those procedures to the SL file, or, if the procedures are present in the group SL or the PUB group SL, run the program with LIB=G or LIB=P.

28 INVALID SYSTEM SL FILE (LOAD ERR 28):

Your SL.PUB.SYS is not a valid SL file. Boy, are you in deep trouble!

WORKAROUND: Do a COLDLOAD or an UPDATE (or, even better, a RELOAD). While you're at it, call your SE, too.

29 INVALID PUBLIC SL FILE (LOAD ERR 29):

The PUB group SL file with which the program was to be loaded is not a valid SL file.

WORKAROUND: Run the program with LIB=S, replace the PUB group SL file with a valid SL file or purge the PUB group SL file. It is usually a good idea to avoid using the name 'SL' for non-SL files.

30 INVALID GROUP SL FILE (LOAD ERR 30):

The group SL file with which the program was to be loaded is not a valid SL file.

WORKAROUND: Run the program with LIB=G or LIB=P, replace the group SL file with a valid SL file or purge the group SL file. It is usually a good idea to avoid using the name 'SL' for SL files.

31 INVALID PROGRAM FILE (LOAD ERR 31):

The file to be loaded is not a valid program file (i.e. does not have filecode PROG).

WORKAROUND: Do not try to run the specified file.

32 INVALID LIST FILE (LOAD ERR 32)

33 CODE SEGMENT TOO LARGE (LOAD ERR 33):

One of the program file's code segments is larger than the maximum permitted by your system configuration.

WORKAROUND: Re-segment the program file or get the system manager to increase the maximum code segment size for your system. Note: The MPE V System Manager/System Supervisor Manual suggests to set this parameter to 5% of available memory; i.e. systems with 768 Kilobytes or more of memory should have this parameter set to 16384 words, the maximum possible.

34 PROGRAM FILE'S EXTENT MAXIMUM MUST BE ONE (LOAD ERR 34):

A program file must have its maximum number of extents equal to 1.

WORKAROUND: Do not run this program file or change its number of extents (possibly with MPEX's %ALTFILE...;EXTENTS= command).

35 DATA SEGMENT TOO LARGE (LOAD ERR 35):

The total amount of stack data segment space (including stack, system overhead and stack expansion area) necessary for this program exceeds the system maximum of 32K.

WORKAROUND: Decrease the amount of stack space used by the program. It is usually a good idea to keep this quantity less than 25 to 30 thousand words.

36 DATA SEGMENT TOO LARGE (LOAD ERR 36):

The total amount of stack data segment space (including stack, system overhead and stack expansion area) necessary for this program exceeds the configuration maximum.

WORKAROUND: Decrease the amount of stack space used by the program or increase the configuration maximum.

37 TOO MANY CODE SEGMENTS (LOAD ERR 37):

The program file to be run contains more than 63 segments, the maximum allowed by the system.

WORKAROUND: Re-segment the program so it will have less than 63 segments or separate the functions of the program into two or more different programs (possibly linked via Process Handling).

38 TOO MANY CODE SEGMENTS (LOAD ERR 38):

The program file to be run contains more code segments than the configuration maximum.

WORKAROUND: Re-segment the program to decrease the number of code segments, separate the functions of the program into two or more different programs (possibly linked via Process Handling) or increase the configuration maximum.

39 ILLEGAL CAPABILITY (LOAD ERR 39):

Either:

- The program to be run has a capability (e.g. PH, PM, MR or DS) that the group in which it resides or the account in which it resides does not have,
- The program is a TEMPORARY FILE and has a capability that the user who's running it doesn't have,
- The program is being run online, but lacks IA capability or is being run offline but lacks BA capability or
- The program contains a privileged segment but lacks PM capability.

WORKAROUND:

- The group or account in which the program resides (not the user running the program!) should be given all the capabilities that the program has or some of the capabilities should be taken away from the program (e.g. with MPEX's %ALTFILE..., CAP= command.
- The user running the program should be given the appropriate capabilities or the program should have capabilities taken away from it or it should be :SAVED and then treated as a temporary file (as in case 1);
- The program should be either given the missing capability (IA or BA) or run in the right mode (offline or online);
- The program should either be given PM capability or have its privileged segments removed.

40 TOO MANY PROCEDURES LOADED (LOAD ERR 40):

An attempt was made to load (via the LOADPROC intrinsic) more than (approximately) 250 procedures.

Appendix C

WORKAROUND: Do not attempt to do more than 250 LOADPROCs. Note: Unloading some of the procedures loaded (via the UNLOADPROC intrinsic) will not help!

41 UNKNOWN PROCEDURE NAME (LOAD ERR 41):

The procedure which was to be loaded (via the LOADPROC intrinsic) does not exist in the SL file(s) from which it was to be loaded.

WORKAROUND: Do not try to load this procedure, add a procedure with this name to the SL file, or, if the procedure already exists in a different SL file, specify a different library search parameter in the LOADPROC invocation.

42 INVALID PROCEDURE NUMBER (LOAD ERR 42)

This message is no longer used.

43 ILLEGAL PROCEDURE UNLOAD (LOAD ERR 43)

This message is no longer used.

44 ILLEGAL SL CAPABILITY (LOAD ERR 44):

The SL with which this program was to be run has privileged procedures, but does not reside in a group and account with PM capability.

WORKAROUND: Remove the privileged procedure from the SL, give PM capability to the group and account in question or do not use this SL.

45 INVALID ENTRY POINT (LOAD ERR 45)

The entry point specified for the program or procedure to be loaded was found, but it pointed to invalid code segment information.

WORKAROUND: The problem is not with the entry point you specified, but is with the program file itself. Recompile the program and try again.

46 TEMPORARY PROGRAM FILE ILLEGAL (LOAD ERR 46)

An attempt was made to load a program that is in the temporary file domain. Only files in the permanent file domain can be allocated.

WORKAROUND: Make the file permanent with the SAVE command, and try again.

50 UNABLE TO OPEN SYSTEM SL FILE (LOAD ERR 50):

The file in question cannot be opened. Possible reasons are that the file is opened exclusively, its file label is clobbered.

WORKAROUND: Do not use the SL file in question; or, determine exactly why the file cannot be opened and take appropriate action.

51 UNABLE TO OPEN PUBLIC SL FILE (LOAD ERR 51):

The file in question cannot be opened. Possible reasons are that the file is opened exclusively, its file label is clobbered.

WORKAROUND: Do not use the SL file in question; or, determine exactly why the file cannot be opened and take appropriate action.

52 UNABLE TO OPEN GROUP SL FILE (LOAD ERR 52):

The file in question cannot be opened. Possible reasons are that the file is opened exclusively, its file label is clobbered.

WORKAROUND: Do not use the SL file in question; or, determine exactly why the file cannot be opened and take appropriate action.

53 UNABLE TO OPEN PROGRAM FILE (LOAD ERR 53):

The file in question cannot be opened. Possible reasons are that the file is opened exclusively, its file label is clobbered.

WORKAROUND: Do not run the program in question or determine exactly why the file cannot be opened and take appropriate action.

54 UNABLE TO OPEN LIST FILE (LOAD ERR 54)

The LOADLIST file cannot be opened. The file might be open exclusively or its file label might be defective.

WORKAROUND: If an asterisk (*) appears after the filename when you execute the command "LISTF *filename*,2", the problem might be that another process has exclusive access to the file. If not, then the file might need to be recreated or the operating system might need to be reloaded.

55 UNABLE TO CLOSE SYSTEM SL FILE (LOAD ERR 55)

A process attempted to close the SL file SL.PUB.SYS, but the close was not successful.

This error condition could indicate a corrupt file or a problem with the operating system. Report this error to your system administrator.

WORKAROUND: Perform an UPDATE/START to put a new copy of the System SL file on your system.

56 UNABLE TO CLOSE PUBLIC SL FILE (LOAD ERR 56)

A process attempted to close the SL file in the PUB group, but the close was not successful.

This error condition could indicate a corrupt file or a problem with the operating system. Report this error to your system administrator.

WORKAROUND: Restore a new copy of the SL from a known good backup.

57 UNABLE TO CLOSE GROUP SL FILE (LOAD ERR 57)

A process attempted to close the group SL file, but the close was not successful.

This error condition could indicate a corrupt file or a problem with the operating system. Report this error to your system administrator.

WORKAROUND: Restore a new copy of the SL from a known good backup.

58 UNABLE TO CLOSE PROGRAM FILE (LOAD ERR 58)

A process attempted to close a program file, but the close was not successful.

This error condition could indicate a corrupt file or a problem with the operating system. Report this error to your system administrator.

WORKAROUND: Recompile or Restore a good copy of the program file.

59 UNABLE TO CLOSE LIST FILE (LOAD ERR 59)

A process attempted to close a list file, but the close was not successful.

This error condition could indicate a corrupt file or a problem with the operating system. Report this error to your system administrator.

60 EOF OR I/O ERROR ON SYSTEM SL FILE (LOAD ERR 60).

The program to be loaded could not be run because the loader got an end-of-file indication or I/O error while reading the specified file. This could mean that 1) the specified file is messed up or 2) the specified file could not be accessed because it is currently being STOREd (this condition often occurs during a SYSDUMP).

WORKAROUND: In case 1, correct the file in question; in case 2, wait until STORE or SYSDUMP unlocks the file or (in case the error occurs on the program file) copy the program file into another file and then run the new file.

61 EOF OR I/O ERROR ON PUBLIC SL FILE (LOAD ERR 61).

The program to be loaded could not be run because the loader got an end-of-file indication or I/O error while reading the specified file. This could mean that 1) the specified file is messed up or 2) the specified file could not be accessed because it is currently being STOREd (this condition often occurs during a SYSDUMP).

WORKAROUND: In case 1, correct the file in question; in case 2, wait until STORE or SYSDUMP unlocks the file or (in case the error occurs on the program file) copy the program file into another file and then run the new file.

62 EOF OR I/O ERROR ON GROUP SL FILE (LOAD ERR 62).

The program to be loaded could not be run because the loader got an end-of-file indication or I/O error while reading the specified file. This could mean that 1) the specified file is messed up or 2) the specified file could not be accessed because it is currently being STOREd (this condition often occurs during a SYSDUMP).

WORKAROUND: In case 1, correct the file in question; in case 2, wait until STORE or SYSDUMP unlocks the file or (in case the error occurs on the program file) copy the program file into another file and then run the new file.

63 EOF OR I/O ERROR ON PROGRAM FILE (LOAD ERR 63).

The program to be loaded could not be run because the loader got an end-of-file indication or I/O error while reading the specified file. This could mean that 1) the specified file is messed up or 2) the specified file could not be accessed because it is currently being STOREd (this condition often occurs during a SYSDUMP).

WORKAROUND: In case 1, correct the file in question; in case 2, wait until STORE or SYSDUMP unlocks the file or (in case the error occurs on the program file) copy the program file into another file and then run the new file.

64 EOF OR I/O ERROR ON LIST FILE (LOAD ERR 64)

The program to be loaded could not be run because the loader got a file system error while reading the list file.

WORKAROUND: The list file might be corrupt or STORE or SYSDUMP might have the file locked. Contact your support representative if problems persist.

65 UNABLE TO OBTAIN CST ENTRIES (LOAD ERR 65)

The loader could not get enough Code Segment Table entries for the program or procedure being loaded.

WORKAROUND: A code segment cannot be larger than 37777 words. An SL can contain no more than 255 segments and a program file can contain no more than 152 segments. Try DEALLOCATEing a procedure not in use.

66 UNABLE TO OBTAIN PROCESS DST ENTRY (LOAD ERR 66)

The loader could not allocate a DST entry for the process being loaded.

WORKAROUND: Try DEALLOCATEing a procedure not in use or increasing the size of the DST.

67 UNABLE TO OBTAIN MAIL DATA SEGMENT (LOAD ERR 67)

This message is no longer used.

68 UNABLE TO CREATE LOAD PROCESS (LOAD ERR 68)

This message is no longer used.

69 UNABLE TO OBTAIN CSTX ENTRIES (LOAD ERR 69)

This message is no longer used.

70 SEGMENT TABLE OVERFLOW (LOAD ERR 70)

The loader was unable to load the program or procedure because the loader segment table is full.

WORKAROUND: Increase the size of the Loader Segment Table if possible or use the DEALLOCATE command to unload unnecessary programs.

71 UNABLE TO OBTAIN SUFFICIENT DL STORAGE (LOAD ERR 71)

The loader could not get enough DL area. This is a system problem.

WORKAROUND: Try to reduce the amount of SL libraries or external procedures being referenced. For example, you might merge group and account SL libraries.

Contact your support representative if this occurs.

72 ATTIO ERROR (LOAD ERR 72):

The loader could not modify the program file's file label.

WORKAROUND: None. The file label may be clobbered or you may be having hardware problems.

73 UNABLE TO OBTAIN VIRTUAL MEMORY (LOAD ERR 73)

Virtual memory was not available when the stack was being created.

WORKAROUND: Try again when the system is not as busy. You may need to increase virtual memory on the system or there might be a hardware problem.

Contact your support representative if problems persist.

74 DIRECTORY I/O ERROR (LOAD ERR 74):

The loader could not retrieve the directory entry of the group in which the program file resides.

WORKAROUND: None. Your directory may be clobbered or you may be having hardware problems.

75 PRINT I/O ERROR (LOAD ERR 75):

The loader got an I/O error while printing a warning (LOAD WARN 89, 90, 91 or 92).

WORKAROUND: None. You may be having hardware problems.

76 ILLEGAL DLSIZE (LOAD ERR 76):

The dlsiz with which the program is to be run exceed 32722 words, the maximum. Note that even this hypothetical maximum is not attainable because dlsizes greater than approximately 27000 words cause the loader to generate a DATA SEGMENT TOO LARGE (LOAD ERR 35).

WORKAROUND: Specify a smaller dlsiz.

77 ILLEGAL MAXDATA (LOAD ERR 77)

This message is no longer used.

80 PROGRAM ALREADY ALLOCATED (LOAD ERR 80):

An attempt was made to allocate a program that was already allocated.

WORKAROUND: None necessary. The program is already allocated.

81 ILLEGAL PROGRAM ALLOCATION (LOAD ERR 81)

An attempt was made to allocate a program that was already allocated.

WORKAROUND: None necessary. The program is already allocated.

82 PROGRAM NOT ALLOCATED (LOAD ERR 82):

An attempt was made to deallocate a program that was not allocated.

WORKAROUND: None necessary. The program is already deallocated.

83 ILLEGAL PROGRAM DEALLOCATION (LOAD ERR 83)

An attempt was made to deallocate a program that was not allocated.

WORKAROUND: None necessary. The program is already deallocated.

84 PROCEDURE ALREADY ALLOCATED (LOAD ERR 84):

An attempt was made to allocate a procedure that was already allocated.

WORKAROUND: None necessary. The procedure is already allocated.

85 ILLEGAL PROCEDURE ALLOCATION (LOAD ERR 85):

Allocation of system procedures (e.g. FOPEN, ATTACHIO, etc.) is not permitted.

WORKAROUND: None.

86 PROCEDURE NOT ALLOCATED (LOAD ERR 86):

An attempt was made to deallocate a procedure that was not allocated.

WORKAROUND: None necessary. The procedure is already deallocated.

87 ILLEGAL PROCEDURE DEALLOCATION (LOAD ERR 87):

Deallocation of system procedures (e.g. FOPEN, ATTACHIO, etc.) is not permitted.

WORKAROUND: None.

88 LMAP NOT AVAILABLE (LOAD WARN 88):

The program was run with a request for an LMAP; however, the program was already loaded — thus the LMAP could not be produced.

WORKAROUND: Do without the LMAP, run the program when it is not loaded or copy the program file into another file and then run it.

89 PROGRAM LOADED WITH LIB = G (LOAD WARN 89):

The program to be run was already loaded with a LIB= parameter other than the one with which it was to be run. This means, for instance, that if you run a program X with LIB=G while it is already loaded with LIB=S (or LIB=P, for that matter), this message will come up and the program will be loaded with the same LIB= parameter as it is already loaded with.

WORKAROUND: Ignore the message, run the program when it is not loaded or copy the program file into another file and then run it.

90 PROGRAM LOADED WITH LIB = P (LOAD WARN 90):

The program to be run was already loaded with a LIB= parameter other than the one with which it was to be run. This means, for instance, that if you run a program X with LIB=G while it is already loaded with LIB=S (or LIB=P, for that matter), this message will come up and the program will be loaded with the same LIB= parameter as it is already loaded with.

WORKAROUND: Ignore the message, run the program when it is not loaded or copy the program file into another file and then run it.

91 PROGRAM LOADED WITH LIB = S (LOAD WARN 91):

The program to be run was already loaded with a LIB= parameter other than the one with which it was to be run. This means, for instance, that if you run a program X with LIB=G while it is already loaded with LIB=S (or LIB=P, for that matter), this message will come up and the program will be loaded with the same LIB= parameter as it is already loaded with.

WORKAROUND: Ignore the message, run the program when it is not loaded or copy the program file into another file and then run it.

92 ATTEMPTING TO ALLOCATE PROGRAM FROM NON-SYSTEM DISC (LOAD ERR 92):

Allocation of programs that reside on private volumes is forbidden.

WORKAROUND: None.

93 UNABLE TO MOUNT PROGRAM FILE'S HOME VOLUME SET (LOAD ERR 93):

The home volume set of the indicated file cannot be mounted.

WORKAROUND: Determine what caused the mount failure and take appropriate action.

94 UNABLE TO MOUNT SYSTEM SL'S HOME VOLUME SET (LOAD ERR 94):

The home volume set of the indicated file cannot be mounted.

WORKAROUND: Determine what caused the mount failure and take appropriate action.

95 UNABLE TO MOUNT PUBLIC SL'S HOME VOLUME SET (LOAD ERR 95):

The home volume set of the indicated file cannot be mounted.

WORKAROUND: Determine what caused the mount failure and take appropriate action.

96 UNABLE TO MOUNT GROUP SL'S HOME VOLUME SET (LOAD ERR 96):

The home volume set of the indicated file cannot be mounted.

WORKAROUND: Determine what caused the mount failure and take appropriate action.

97 UNABLE TO LOAD REMOTE PROGRAM FILE (LOAD ERR 97):

This message is no longer used.

98 UNABLE TO CONVERT OLD FORMAT (LOAD ERR 98):

This message is no longer used.

99 UNABLE TO OBTAIN DST FOR LOGICAL MAP (LOAD ERR 99):

The loader could not get a Data Segment Table entry for the logical map (LSTT).

WORKAROUND: Try DEALLOCATEing unused programs and procedures. You may need to increase the DST size or the maximum configuration for extra data segment. Virtual memory size might also need to be increased.

100 TOO MANY MAPPED SEGMENTS (LOAD ERR 100):

Too many user SL segments were referenced. Total user SL segments cannot exceed 254.

WORKAROUND: Reduce the amount of segments in the user SL. Try combining some of the user SL segments for this program.

101 SEGMAP TOO BIG (LOAD ERR 101):

The loader was unable to allocate space in the logical map (LSTT). The logical map is full.

WORKAROUND: Reduce the amount of segments in the user SL. Try combining some of the user SL segments for this program.

102 UNABLE TO EXPAND SEGMAP (LOAD ERR 102):

The loader was trying to allocate a larger data segment for the logical map (LSTT) when an error occurred.

WORKAROUND: Try again when the system is not as busy. Virtual memory might need to be increased or there might be a hardware problem. Contact your support representative if problems persist.

103 TOO MANY DYNAMIC LOADS ON PROCEDURE (LOAD ERR 103):

A single process has attempted to load a procedure more than 65,535 times. This is probably a programming error since usually a procedure is loaded and then unloaded when not needed.

WORKAROUND: Change the program logic to call UNLOADPROC to release the procedure when it is no longer needed. Check for conditions that would load a procedure that was already loaded.

104 UNABLE TO OBTAIN DATA SEGMENT. COMMAND IGNORED (LOAD ERR 104):

The SHOWALLOCATE command was not able to obtain a temporary extra data segment.

WORKAROUND: Try DEALLOCATEing unnecessary programs and procedures. Increase the DST size or the maximum configuration for extra data segment size. Virtual memory size might also need to be increased.

105 UNABLE TO OPEN CONFDATA.PUB.SYS. COMMAND IGNORED (LOAD ERR 105):

This message is no longer used.

106 EOF OR I/O ERROR ON CONFDATA.PUB.SYS. COMMAND IGNORED (LOAD ERR 106):

This message is no longer used.

107 UNABLE TO CLOSE CONFDATA.PUB.SYS. COMMAND IGNORED (LOAD ERR 107):

This message is no longer used.

108 EOF OR I/O ERROR ON SHOWALLOCATE LIST FILE. COMMAND ABORTED (LOAD ERR 108):

The SHOWALLOCATE command had trouble using the list file specified with the command or with \$STDLIST if no file was specified.

WORKAROUND: If you specified a file for output from this command, you might need to make it larger. If it is large enough, the file label might be corrupt. Check hardware and verify file label integrity. Contact your support representative if problems persist.

109 I/O ERROR OCCURRED WHILE READING FILE LABEL. COMMAND ABORTED (LOAD ERR 109):

The loader had a problem reading from the program file label.

WORKAROUND: The file label might be corrupt. Recompile the program or restore it from a backup tape. If this does not solve the problem, you might have a hardware problem or you might need to reload the operating system by doing a COLDLOAD. Contact your support representative if problems persist.

Appendix D: CRYPTIC FILE SYSTEM ERROR MESSAGES DECRYPTED

In addition to its other failings, the System Intrinsic Manual does not explain the exact reason for and/or workaround for most file system errors. In fact, most file system error messages are very hard to understand.

The following is an attempt at an adequate explanation of the causes, effects and workarounds for different file system errors that pertain to disc files:

0 END OF FILE (FSERR 0):

This error is encountered when a program attempts to read beyond the end of file or write beyond the file limit.

WORKAROUND: Change the program or the file.

1 ILLEGAL DB REGISTER SETTING (FSERR 1):

Should never occur for non-privileged mode programs. For privileged mode programs, this means that the programmer attempted to do an FFILEINFO, FGETINFO, FOPEN or FRENAME in split-stack mode (i.e. after a call to the EXCHANGEDB or SWITCHDB procedures).

WORKAROUND: Do not perform the function in split-stack mode.

2 ILLEGAL CAPABILITY (FSERR 2):

A function that requires privileged mode capability (e.g. open a file for NOWAIT I/O, open a file for EXECUTE access, etc.) was attempted without privileged mode capability.

WORKAROUND: Enter privileged mode before executing the function or do not attempt to execute it at all.

3 REQUIRED PARAMETER IS MISSING (FSERR 3):

4 DISC FREE SPACE ALLOCATION DISABLED ON ALL DISCS IN DOMAIN (FSERR 4):

5 DRT NUMBER > 511 (FSERR 5):

6 DEVICE HAS NO AVAILABLE SPARE BLOCKS (FSERR 6):

7 UNFORMATTED OR UNINITIALIZED MEDIA ON DEVICE. (FSERR 7):

8 ILLEGAL PARAMETER VALUE (FSERR 8):

Parameters specified in FOPEN call are mutually contradictory; for instance, an attempt to open a file NOWAIT on a serial disc was detected or the program tried to open a new KSAM file without specifying the FORMALDESIGNATOR or KSAMPARAM parameters on the FOPEN.

WORKAROUND: Correct the parameter.

9 INVALID FILE TYPE SPECIFIED IN FOPTIONS (FSERR 9):

The file type field of the FOPEN file options is not one of 0 (STD = standard file), 1 (KSAM file), 2 (RIO file), 4 (CIR = circular file) or 6 (MSG = message file).

WORKAROUND: Correct the file type field.

10 INVALID RECORD SIZE SPECIFICATION (FSERR 10):

The record size requested was more than 32,767 bytes.

Appendix D

WORKAROUND: Specify a smaller record size.

11 INVALID RESULTANT BLOCK SIZE (FSERR 11):

If the user requests were honored, the block size (BLOCK FACTOR * RECORD SIZE) of the resultant file would be greater than 32,767 bytes.

WORKAROUND: Specify a smaller record size or blocking factor.

12 RECORD NUMBER OUT OF RANGE (FSERR 12):

The user passed a negative record number to the FPOINT, FREADDIR or FWRITEDIR intrinsic — this is illegal.

WORKAROUND: Correct your program.

13 CAN'T OPEN FILE MULTI-ACCESS, OUT OF FMAVT ENTRIES (FSERR 13):

15 EXCEEDED THE MAXIMUM MESSAGE FILE OPENS FOR WRITE ACCESS (FSERR 15):

16 MORE THAN 255 OPENS OF A FILE (FSERR 16):

17 MAGNETIC TAPE RUNAWAY (FSERR 17):

18 DEVICE POWERED UP (FSERR 18):

19 FORMS CONTROL WAS RESET (FSERR 19):

20 INVALID OPERATION (FSERR 20):

21 DATA PARITY ERROR (FSERR 21):

22 SOFTWARE TIME-OUT (FSERR 22):

The user tried to read an empty message file or write to a full message file, an action which would cause the user to be impeded until the file was no longer empty or full, respectively (see MPE IV INTRINSICS MANUAL). However, a time out was set with the FCONTROL intrinsic (mode 4) and the request timed out before it could be honored.

WORKAROUND: Do not set the time out or ensure that the request can be serviced before it times out.

23 END OF TAPE (FSERR 23):

24 UNIT NOT READY (FSERR 24):

25 NO WRITE-RING ON TAPE (FSERR 25):

26 TRANSMISSION ERROR (FSERR 26):

Hardware failure.

WORKAROUND: Call your CE.

27 I/O TIME-OUT (FSERR 27):

28 TIMING ERROR OR DATA OVERRUN (FSERR 28):

29 SIO FAILURE (FSERR 29):

30 UNIT FAILURE (FSERR 30):

Hardware failure.

WORKAROUND: Call your CE.

31 END OF LINE (FSERR 31):

32 SOFTWARE ABORT (FSERR 32):

33 DATA LOST (FSERR 33):

34 UNIT NOT ON-LINE (FSERR 34):

35 DATA-SET NOT READY (FSERR 35):

36 INVALID DISC ADDRESS (FSERR 36):

37 INVALID MEMORY ADDRESS (FSERR 37):

38 TAPE PARITY ERROR (FSERR 38):

39 RECOVERY TAPE ERROR (FSERR 39):

40 OPERATION INCONSISTENT WITH ACCESS TYPE (FSERR 40):

The access type specified at FOPEN time does not permit this operation; for instance, an FWRITE is not permitted when a file is opened with ACC=IN.

WORKAROUND: Specify an access type at FOPEN time which permits this operation or do not perform the operation at all.

41 OPERATION INCONSISTENT WITH RECORD TYPE (FSERR 41):

It seems that this error should never show up and is merely a leftover from a previous version of MPE.

42 OPERATION INCONSISTENT WITH DEVICE TYPE (FSERR 42):

The program tried to execute an operation that is incompatible with the device that it is trying to perform it on; for instance, it is trying to read the line printer or change the baud rate of a disk drive.

WORKAROUND: Do not execute the operation.

43 WRITE EXCEEDS RECORD SIZE (FSERR 43):

An attempt was made to write a record that would not fit in the destination file, e.g. trying to write a 100-byte record into a file with a record length of 80 bytes.

WORKAROUND: Change the file's record size, change the length of the record to be written or open the file with the Multi-Record (MR) access option.

44 UPDATE AT RECORD ZERO (FSERR 44):

The FUPDATE intrinsic (which is equivalent to the COBOL REWRITE statement) was called with the record pointer at record 0, which indicates that no record has been read and therefore no record can be updated.

WORKAROUND: Call FPOINT or FREAD before the FUPDATE call.

45 PRIVILEGED FILE VIOLATION (FSERR 45):

A program attempted to open a privileged file (one with a negative file code; e.g. an IMAGE file) while specifying a filecode not equal to the file's filecode or while not in privileged mode.

WORKAROUND: Enter privileged mode before the call or specify the correct filecode.

46 OUT OF DISC SPACE (FSERR 46):

The device class on which this file resides (if this error occurs at extent allocation time) or is requested to reside (if this error occurs at file creation time) does not have enough contiguous disk space to accommodate this file; i.e. if NUMEXTS is the number of extents to be allocated and EXTSIZE is the size (in sectors) of one extent, this device class does not have NUMEXTS contiguous chunks of EXTSIZE sectors each.

WORKAROUND: Move the file to another, less full, device class, decrease the requested file size or decrease the extent size by increasing the number of extents in the file.

47 I/O ERROR ON FILE LABEL (FSERR 47):

The internal file label of this file cannot be accessed. Most likely, the file is totally clobbered and will return INVALID FILE LABEL (FSERR 108) when it is subsequently accessed.

WORKAROUND: None.

48 OPERATION INVALID DUE TO MULTIPLE FILE ACCESS (FSERR 48):

One of the following conditions is true: 1) The program is trying to purge (i.e. close with disposition DEL) a file that is currently loaded or being stored/restored, 2) The program is trying to rename (with the FRENAME intrinsic) a file that it does not have exclusive access to or 3) The program is trying to open with LOCK access a file that someone else has opened with NOLOCK access or vice versa.

WORKAROUND: 1) Don't purge the file or wait for the file to become purgeable again, 2) Don't rename the file or open the file with EXC access or 3) Open the file with LOCK or NOLOCK access (whichever the other program has the file open with).

49 UNIMPLEMENTED FUNCTION (FSERR 49):

The program specified an invalid parameter value in a file system intrinsic call; e.g. a disposition of 5, 6 or 7 at FCLOSE time or a file type of RIO on pre-Athena systems (ones which do not support RIO files).

WORKAROUND: Correct your program.

50 NONEXISTENT ACCOUNT (FSERR 50):

An attempt was made to open a file in an account which was not configured in the system.

WORKAROUND: Correct the filename or build the account.

51 NONEXISTENT GROUP (FSERR 51):

An attempt was made to open a file in a group which was not configured in the system.

WORKAROUND: Correct the filename or build the group.

52 NONEXISTENT PERMANENT FILE (FSERR 52):

An attempt was made to open a file which does not exist.

WORKAROUND: Correct the program or build the file.

53 NONEXISTENT TEMPORARY FILE (FSERR 53):

The program tried to open a temporary file which does not exist.

WORKAROUND: Correct the program or build the file.

54 INVALID FILE REFERENCE (FSERR 54):

The program tried to open a file whose filename was invalid; for instance, the file, group or account name was longer than 8 characters, an invalid system-defined file was specified (e.g. \$XYZZY) or no file equation was found for a back-referenced file (e.g. *MANSION with no file equation for file MANSION).

WORKAROUND: Correct the filename specified.

55 DEVICE UNAVAILABLE (FSERR 55):

The program tried to open a message file that was already opened for MULTI (not GMULTI) access by another job/session.

WORKAROUND: Wait for the other job/session to finish or rewrite the other program to open the file NOMULTI or GMULTI.

56 INVALID DEVICE SPECIFICATION (FSERR 56):

The device number or device class on which the file was to be opened is not configured on the system.

WORKAROUND: Correct the program.

57 OUT OF VIRTUAL MEMORY (FSERR 57):

The buffer size (NUMBER OF BUFFERS * RECORD SIZE * BLOCKING FACTOR) of the file to be opened exceeds 8,192 words (or 14,000 words starting with the D-MIT version of MPE IV).

WORKAROUND: Decrease the number of buffers (by specifying BUF=1 on a :FILE equation, for instance), decrease the record size of the file or decrease the blocking factor of the file.

58 NO PASSED FILE (FSERR 58):

The program attempted to open \$OLDPASS, but no \$OLDPASS file exists.

WORKAROUND: Correct the program or build a \$OLDPASS file.

60 GLOBAL RIN UNAVAILABLE (FSERR 60):

The program requested dynamic locking at file open time, but the RIN (Resource Identification Number) necessary for dynamic locking could not be obtained.

WORKAROUND: Free some global RINs (with the :FREERIN command), file RINs (by closing files opened with LOCK access), open the file with NOLOCK access or enlarge the RIN table.

61 OUT OF GROUP DISC SPACE (FSERR 61):

The program tried to allocate more disk space than is allowed for a given group; e.g. it tried to build a 10,000-sector file in a group which already had 95,000 sectors and was limited to 100,000 sectors.

WORKAROUND: Decrease the amount of disk space used by files in that group (by purging or squeezing files) or ask the account manager to increase the group disk space limit.

62 OUT OF ACCOUNT DISC SPACE (FSERR 62):

The program tried to allocate more disk space than is permitted for the account in which it tried to allocate it.

WORKAROUND: Decrease the amount of disk space used by files in that account (by purging or squeezing files) or ask the system manager to increase the account disk space limit.

64 USER LACKS MULTI-RIN CAPABILITY (FSERR 64):

The program was not :PREPped with MR (Multi-Rin) capability, yet tried to lock a file when another file (or RIN) was already locked by that program.

WORKAROUND: :PREP the program with MR capability or do not try to lock a file when you have already locked another one.

65 PUNCH HOPPER EMPTY (FSERR 65):

66 PLOTTER LIMIT SWITCH REACHED (FSERR 66):

67 PAPER TAPE ERROR (FSERR 67):

68 INSUFFICIENT SYSTEM RESOURCES (FSERR 68):

69 I/O ERROR (FSERR 69):

70 I/O ERROR WHILE PRINTING HEADER/TRAILER (FSERR 70):

71 TOO MANY FILES OPEN (FSERR 71):

The program attempted to open a file, but there was not enough room in the system area (PCBX) of the program's stack for the information for that file.

WORKAROUND: Close some files which are no longer necessary before trying the open or run the program with the ;NOCB keyword in the :RUN command.

72 INVALID FILE NUMBER (FSERR 72):

An attempt was made to access (e.g. read or write) a file that had not been opened or that is a privileged file; for instance, a read was requested against file number 10, but no file number 10 was open.

WORKAROUND: Correct your program or enter privileged mode before trying to access the file (if the file is privileged).

73 BOUNDS VIOLATION (FSERR 73):

You are attempting to read or write more data than could fit into your I/O buffer (e.g. you are trying to read 100 words into an 80-word array).

WORKAROUND: Decrease the length of the data you are trying to read or write or enlarge your program's I/O buffer.

74 NO ROOM LEFT IN STACK SEGMENT FOR ANOTHER FILE ENTRY (FSERR 74):

When trying to do an FOPEN, the file system tried to expand your stack data segment to have room for the information on the file being opened; this caused an error because it would make the data segment too large.

WORKAROUND: Often, this does not mean that you were using too much stack space AT THE TIME OF THE FOPEN. Rather, you might have used a lot of stack space earlier and that space was not deallocated (since your stack data segment is never automatically shrunk by the system). If you call "ZSIZE (0)" (to shrink the stack data segment to only the amount you're currently using) immediately before the FOPEN call, the problem may very well go away. If it doesn't, you might consider using less stack space or running your program with ;NOCB.

77 NO-WAIT I/O PENDING (FSERR 77):

78 NO NO-WAIT I/O PENDING FOR ANY FILE (FSERR 78):

79 NO NO-WAIT I/O PENDING FOR SPECIAL FILE (FSERR 79):

80 SPOOFLE SIZE EXCEEDS CONFIGURATION (FSERR 80):

81 NO "SPOOL" CLASS IN SYSTEM (FSERR 81):

82 INSUFFICIENT SPACE FOR SPOOFLE (FSERR 82):

83 I/O ERROR ON SPOOFLE (FSERR 83):

84 DEVICE UNAVAILABLE FOR SPOOFLE (FSERR 84):

85 OPERATION INCONSISTENT WITH SPOOLING (FSERR 85):

86 SPOOLING INTERNAL ERROR (FSERR 86):

87 BAD SPOOFLE BLOCK (FSERR 87):

88 NONEXISTENT SPOOFLE (FSERR 88):

89 POWER FAILURE (FSERR 89):

90 EXCLUSIVE VIOLATION: FILE BEING ACCESSED (FSERR 90):

Exclusive access was requested to a file which is already being accessed; thus, exclusive access cannot be granted.

WORKAROUND: Specify SHR (shared) or EAR (exclusive - allow read) access when opening the file or wait for the accessor to close the file.

91 EXCLUSIVE VIOLATION: FILE BEING ACCESSED EXCLUSIVELY (FSERR 91):

Access was requested to a file which is being accessed exclusively by some other user.

WORKAROUND: Wait for the accessor to close the file.

92 LOCKWORD VIOLATION (FSERR 92):

An invalid lockword was specified at file open time or when the file system prompted the user for a lockword.

WORKAROUND: Specify a correct lockword or remove or change the lockword on the disc file.

93 SECURITY VIOLATION (FSERR 93):

Permitting the user to access this file in the specified access mode would be a breach of file security.

WORKAROUND: Change the access mode specified in the program to one which is permitted or ask the file's creator to :RELEASE or :ALTSEC the file.

94 USER IS NOT CREATOR (FSERR 94):

An attempt was made to :RENAME or FRENAME a file by someone other than the file's creator.

WORKAROUND: Do not perform the :RENAME or FRENAME, ask the creator of the file to do the :RENAME or (if you have read and write access to the file and are a user of MPEX) use MPEX's %RENAME command.

95 READ COMPLETED DUE TO BREAK (FSERR 95):

96 DISC I/O ERROR (FSERR 96):

Hardware failure. Call your CE.

Appendix D

97 NO CONTROL Y PIN (FSERR 97):

98 READ TIME OVERFLOW (FSERR 98):

99 BOT AND BACKSPACE TAPE (FSERR 99):

100 DUPLICATE PERMANENT FILE NAME (FSERR 100):

The program tried to save (close with SAVE disposition) a new or temporary file as a permanent file, but a permanent file with that name already exists.

WORKAROUND: Purge the other file with that name.

101 DUPLICATE TEMPORARY FILE NAME (FSERR 101):

The program tried to save as a temporary file (close with TEMP disposition) a new file, but a temporary file with that name already exists.

WORKAROUND: Purge the other temporary file with that name.

102 DIRECTORY I/O ERROR (FSERR 102):

The directory (or part of it) is clobbered. You're in big trouble.

WORKAROUND: None.

103 PERMANENT FILE DIRECTORY OVERFLOW (FSERR 103):

There is no more room in the system file directory for this file (the system file directory typically allows approximately 1,200 files per group).

WORKAROUND: Purge some of the files in the group in which you wish to build the file.

104 TEMPORARY FILE DIRECTORY OVERFLOW (FSERR 104):

There is no more room in your job / session temporary file directory for this file.

WORKAROUND: Purge some temporary files or :RESET some :FILE equations or :CRESET some :CLINE equations.

105 BAD VARIABLE BLOCK STRUCTURE (FSERR 105):

The variable record length file being accessed has an inconsistent structure or would have an inconsistent structure if this access were to go through (if you are writing NOBUF).

WORKAROUND: If you are writing NOBUF, correct your program; otherwise, none.

106 EXTENT SIZE EXCEEDS MAXIMUM (FSERR 106):

The program attempted to build a file which would have extents larger than 65,534 sectors, the maximum permitted.

WORKAROUND: Increase the number of extents in the file or decrease the extent size by decreasing the record size or file limit of the file.

107 INSUFFICIENT SPACE FOR USER LABELS (FSERR 107):

The maximum number of user labels for a file is 254.

WORKAROUND: Decrease the number of user labels requested by the program.

108 INVALID FILE LABEL (FSERR 108):

The file is inaccessible because the file is invalid (probably destroyed - and unrecoverable).

WORKAROUND: None.

109 INVALID CARRIAGE CONTROL (FSERR 109):

The program tried to do a write with a CCTL code of 1 (embedded CCTL) but with a buffer length of 0; or, the program attempted an FCONTROL mode 1 (transfer CCTL code) with a parameter of 1.

WORKAROUND: Correct the program.

110 ATTEMPT TO SAVE PERMANENT FILE AS TEMPORARY (FSERR 110):

An attempt was made to close a permanent file with temporary (TEMP) disposition: this is illegal.

WORKAROUND: Correct the program.

111 USER LACKS SAVE FILES (SF) CAPABILITY (FSERR 111):**112 USER LACKS PRIVATE VOLUMES (UV) CAPABILITY (FSERR 112):****113 VOLUME SET NOT MOUNTED - MOUNT PROBLEM (FSERR 113):****114 VOLUME SET NOT DISMOUNTED - DISMOUNT PROBLEM (FSERR 114):****115 ATTEMPTED RENAME ACROSS VOLUME SETS - REJECTED (FSERR 115):****116 INVALID TAPE LABEL FOPEN PARAMETERS (FSERR 116):****117 ATTEMPTED TO WRITE ON AN UNEXPIRED TAPE FILE (FSERR 117):****118 INVALID HEADER OR TRAILER TAPE LABEL (FSERR 118):****119 I/O ERROR POSITIONING TAPE FOR TAPE LABELS (FSERR 119):****120 ATTEMPT TO WRITE IBM STANDARD TAPE LABEL (FSERR 120):****121 TAPE LABEL LOCKWORD VIOLATION (FSERR 121):****122 TAPE LABEL TABLE OVERFLOW (FSERR 122):****123 END OF TAPE VOLUME SET (FSERR 123):****124 ATTEMPTED TO APPEND LABELED TAPE (FSERR 124):****125 EXPIRATION DATE CAN'T BE LATER THAN PRECEDING FILE'S (FSERR 125):****126 CHARACTER SET NUMBER MUST BE BETWEEN 0 AND 31. (FSERR 126):****127 FORM NUMBER MUST BE BETWEEN 0 AND 31. (FSERR 127):****128 LOGICAL PAGE NUMBER MUST BE BETWEEN 0 AND 31. (FSERR 128):****129 VERTICAL FORMAT NUMBER MUST BE BETWEEN 0 AND 31. (FSERR 129):****130 NUMBER OF COPIES MUST BE BETWEEN 1 AND 32767. (FSERR 130):****131 NUMBER OF OVERLAYS MUST BE BETWEEN 1 AND 8. (FSERR 131):****132 PAGE LENGTH PARM MUST BE BETWEEN 12 (=3") AND 68 (=17"). (FSERR 132):****133 PICTURE NUMBER MUST BE BETWEEN 0 AND 31. (FSERR 133):****134 EXTENDED CAPABILITY PARM MUST BE 0 (OFF) OR 1 (ON). (FSERR 134):****135 UNRECOVERED MEDIA ERROR. (FSERR 135):**

Appendix D

136 USER LIMIT EXCEEDED. (FSERR 136):

137 DEFECTIVE TRACK ON FOREIGN DISC (FSERR 137):

138 TRACK DOES NOT EXIST ON FOREIGN DISC (FSERR 138):

139 DELETED RECORD ON IBM DISKETTE (FSERR 139):

140 ACCESS DENIED DUE TO CORRUPT ACD, FILE OWNER MUST REAPPLY ACD (FSERR 140):

148 INACTIVE RIO RECORD (FSERR 148):

An FPOINT, FREADDIR or FSPACE positioned the record pointer at an inactive record in an RIO (Relative I/O) file.

WORKAROUND: None necessary.

149 MISSING ITEM NUMBER OR RETURN-VARIABLE (FSERR 149):

An item number was specified without a corresponding variable or vice versa in an FFILEINFO intrinsic call.

WORKAROUND: Correct the program.

150 INVALID ITEM NUMBER (FSERR 150):

An item number specified in an FFILEINFO intrinsic call is invalid.

WORKAROUND: Correct the program.

151 CURRENT RECORD WAS LAST RECORD WRITTEN BEFORE SYSTEM CRASHED (FSERR 151):

152 UNRECOGNIZED KEYWORD IN FOPEN DEVICE PARAMETER (FSERR 152):

153 EXPECTED ";" OR CARRIAGE RETURN IN DEVICE PARAMETER (FSERR 153):

154 ENVIRONMENT FILE OPEN ERROR (FSERR 154):

155 NOT ENVIRONMENT FILE.CHECK FILE CODE OR RECORD SIZE. (FSERR 155):

156 ENVIRONMENT HEADER RECORD INCORRECT (FSERR 156):

157 UNCOMPILED ENVIRONMENT FILE (FSERR 157):

158 ERROR READING ENVIRONMENT FILE (FSERR 158):

159 ERROR CLOSING ENVIRONMENT FILE (FSERR 159):

160 ERROR DOING FDEVICECONTROL FROM ENVIRONMENT FILE (FSERR 160):

161 TOO MANY PARAMETERS IN DEVICE STRING - OVERFLOW (FSERR 161):

162 EXPECTED "=" AFTER KEYWORD IN DEVICE PARAMETER (FSERR 162):

163 "ENV" BACK REFERENCE IN FILE EQUATION INCORRECT (FSERR 163):

164 DEVICE PARAMETER TOO LARGE OR MISSING CARRIAGE RETURN (FSERR 164):

165 INVALID DENSITY SPECIFICATION (FSERR 165):

166 FFILEINFO FAILED IN ACCESSING REMOTE SPOOL FILE. (FSERR 166):

167 SPOOL FILE LABEL ERROR, CAN'T INSERT INV FILE NAME. (FSERR 167):

168 ITEM NOT SUPPORTED ON REMOTE SYSTEM. (FSERR 168):

170 RECORD IS MARKED FOR DELETION (FSERR 170):
171 DUPLICATE KEY VALUE (FSERR 171):
172 NO SUCH KEY (FSERR 172):
173 TCOUNT PARAMETER LARGER THAN RECORD SIZE (FSERR 173):
174 CAN NOT GET EXTRA DATA SEGMENT (FSERR 174):
175 KSAM INTERNAL ERROR (FSERR 175):
176 ILLEGAL EXTRA DATA SEGMENT LENGTH (FSERR 176):
177 TOO MANY EXTRA DATA SEGMENTS FOR THIS PROCESS (FSERR 177):
178 NOT ENOUGH VIRTUAL MEMORY FOR EXTRA DATA SEGMENT (FSERR 178):
179 THE FILE MUST BE LOCKED BEFORE ISSUING THIS INTRINSIC (FSERR 179):
180 KSAM FILE MUST BE REBUILT (FSERR 180):
181 INVALID KEY STARTING POSITION (FSERR 181):
182 FILE IS EMPTY (FSERR 182):
183 RECORD DOES NOT CONTAIN ALL KEYS (FSERR 183):
184 INVALID RECORD NUMBER (FFINDN INTRINSIC) (FSERR 184):
185 SEQUENCE ERROR IN PRIMARY KEY (FSERR 185):
186 INVALID KEY LENGTH (FSERR 186):
187 INVALID KEY SPECIFICATION (FSERR 187):
188 INVALID DEVICE SPECIFICATION (FSERR 188):
189 INVALID RECORD FORMAT (FSERR 189):
190 INVALID KEY BLOCKING FACTOR VALUE (FSERR 190):
191 RECORD DOES NOT CONTAIN SEARCH KEY SPECIFIED FOR DELETION (FSERR 191):
192 SYSTEM FAILURE OCCURRED WHILE THE KSAM FILE WAS OPENED (FSERR 192):
193 \$STDIN/\$STDLIST CANNOT BE REDIRECTED TO KSAM FILES (FSERR 193):
194 KSAM FILES NOT ALLOWED FOR GLOBAL AFT'S (FSERR 194):
195 GLOBAL FILES CANNOT BE REMOTE FILES (FSERR 195):
196 LANGUAGE NOT SUPPORTED (FSERR 196):
197 NATIVE LANGUAGE INTERNAL ERROR (FSERR 197):
198 INVALID VERSION NUMBER IN KSAM FILE (FSERR 198):
199 UNABLE TO SWITCH THE LOCKING MODE (FSERR 199):
201 REMOTE DID NOT RESPOND WITH THE CORRECT REMOTE ID. (DSERR 201):
202 SPECIFIED PHONE NUMBER IS INVALID (DSERR 202):

Appendix D

- 203 REMOTE ABORT/RESUME NOT VALID WHEN DOING PROGRAM-TO-PROGRAM COMMUNICATION. USE LOCAL ABORT/RESUME. (DSWARN 203):
- 204 UNABLE TO ALLOCATE AN EXTRA DATA SEGMENT FOR DS/3000. (DSERR 204):
- 205 UNABLE TO EXPAND THE DS/3000 EXTRA DATA SEGMENT. (DSERR 205):
- 206 SLAVE PTOP FUNCTION ISSUED FROM A MASTER PROGRAM. (DSERR 206):
- 207 SLAVE PTOP FUNCTION OUT OF SEQUENCE. (DSERR 207):
- 208 MASTER PTOP FUNCTION ISSUED BY A SLAVE PROGRAM. (DSERR 208):
- 209 SLAVE PROGRAM DOES NOT EXIST OR IS NOT PROGRAM FILE. (DSERR 209):
- 210 WARNING -- INVALID MAXDATA OR DLSIZE FOR A SLAVE PROGRAM.
SYSTEM DEFAULTS ARE IN EFFECT. (DSWARN 210):
- 211 SLAVE ISSUED A REJECT TO A MASTER PTOP OPERATION. (DSWARN 211):
- 212 FILE NUMBER FROM IOWAIT NOT A DS LINE NUMBER. (DSWARN 212):
- 213 EXCLUSIVE USE OF A DS LINE REQUIRES BOTH ND AND CS CAPABILITY. (DSERR 213):
- 214 DS LINE WAS NOT OPENED WITH A USER:DSLIME COMMAND. (DSERR 214):
- 215 DSLIME CANNOT BE ISSUED BACK TO THE MASTER COMPUTER. (DSERR 215):
- 216 MESSAGE REJECTED BY THE REMOTE COMPUTER. (DSERR 216):
- 217 INSUFFICIENT AMOUNT OF USER STACK AVAILABLE. (DSERR 217):
- 218 INVALID PTOP FUNCTION REQUESTED. (DSERR 218):
- 219 MULTIPLE POPEN.
ONLY ONE MASTER PTOP OPERATION CAN BE ACTIVE ON A DSLIME. (DSERR 219):
- 220 PROGRAM EXECUTING GET WAS NOT CREATED BY POPEN. (DSERR 220):
- 221 INVALID DS MESSAGE FORMAT. INTERNAL DS ERROR. (DSERR 221):
- 222 MASTER PTOP FUNCTION ISSUED PRIOR TO A POPEN. (DSERR 222):
- 223 REQUEST TO SEND MORE DATA THAN SPECIFIED IN POPEN (DSERR 223):
- 224 FILE EQUATIONS FOR A REMOTE FILE CONSTITUTE A LOOP. (DSERR 224):
- 225 CANNOT ISSUE POPEN TO A SLAVE SESSION IN BREAK MODE. (DSERR 225):
- 226 SLAVE PROGRAM HAS TERMINATED BEFORE EXECUTING "GET". (DSERR 226):
- 227 RFA/RDBA COULD NOT FIND OR CREATE A REMOTE SESSION. (DSERR 227):
- 228 EXCEEDED MAXIMUM NUMBER OF VIRTUAL CHANNELS PER JOB. (DSERR 228):
- 229 TOO MANY DS USERS ON THIS SYSTEM.
CANNOT OBTAIN A VIRTUAL PROCESS NUMBER. (DSERR 229):
- 230 (DSERR 230):
- 231 INVALID FACILITY IN CONNECTION REQUEST. (DSERR 231):

232 THE REMOTE COMPUTER IS NOT OBTAINABLE. (DSERR 232):

233 VIRTUAL CIRCUIT IS NOT AVAILABLE. (DSERR 233):

234 QUEUEING IS REQUIRED TO COMPLETE THE REQUEST. (DSERR 234):

235 DS MESSAGE SEQUENCING ERROR. (DSERR 235):

236 COMMUNICATIONS HARDWARE HAS DETECTED AN ERROR. (DSERR 236):

237 CANNOT CURRENTLY GAIN ACCESS TO THE TRACE FILE. (DSERR 237):

238 COMMUNICATIONS INTERFACE ERROR. INTERNAL FAILURE. (DSERR 238):

239 COMMUNICATIONS INTERFACE ERROR. TRACE MALFUNCTION. (DSERR 239):

240 LOCAL COMMUNICATION LINE WAS NOT OPENED BY OPERATOR. (DSERR 240):

241 DS LINE IN USE EXCLUSIVELY OR BY ANOTHER SUBSYSTEM (DSERR 241):

242 INTERNAL DS SOFTWARE ERROR ENCOUNTERED. (DSERR 242):

243 REMOTE OR PDN IS NOT RESPONDING. (DSERR 243):

244 COMMUNICATIONS INTERFACE ERROR. LINE RESET OCCURRED. (DSERR 244):

245 COMMUNICATIONS INTERFACE ERROR. RECEIVE TIMEOUT. (DSERR 245):

246 COMMUNICATIONS INTERFACE ERROR. REMOTE DISCONNECTED. (DSERR 246):

247 COMMUNICATIONS INTERFACE ERROR. LOCAL TIME OUT. (DSERR 247):

248 COMMUNICATIONS INTERFACE ERROR. CONNECT TIME OUT. (DSERR 248):

249 COMMUNICATIONS INTERFACE ERROR. REMOTE REJECTED CONNECTION. (DSERR 249):

250 COMMUNICATIONS INTERFACE ERROR. CARRIER LOST. (DSERR 250):

251 COMMUNICATIONS INTERFACE ERROR.
LOCAL DATA SET FOR THE DS LINE WENT NOT READY. (DSERR 251):

252 COMMUNICATIONS INTERFACE ERROR. HARDWARE FAILURE. (DSERR 252):

253 COMMUNICATIONS INTERFACE ERROR.
NEGATIVE RESPONSE TO THE DIAL REQUEST BY THE OPERATOR. (DSERR 253):

254 COMMUNICATIONS INTERFACE ERROR. INVALID I/O CONFIGURATION. (DSERR 254):

255 COMMUNICATIONS INTERFACE ERROR. UNANTICIPATED CONDITION. (DSERR 255):

257 DATA TRANSFER COUNT TOO LARGE. (DSERR 257):

300 NO. OF OPENS FOR FILE EXCEEDS 255 (FSERR 300):

301 FREE SPACE TABLE FOR LDEV ! IS FULL, RUN VINIT -COND (FSERR 301):

302 INVALID ITEM NUMBER FOR FDEVICECONTROL (FSERR 302):

303 INVALID ACCESS FOR ITEM NUMBER TO FDEVICECONTROL (FSERR 303):

304 ATTEMPT TO CHANGE TERMINAL PARITY IN 8 BIT MODE (FSERR 304):

305 INVALID FORMAT IN TERMINAL CONFIGURATION FILE (FSERR 305):

Appendix D

306 CHECKSUM ERROR IN TERMINAL CONFIGURATION FILE (FSERR 306):
307 PASSED VALUE TO FDEVICECONTROL LESS THAN MIN (FSERR 307):
308 PASSED VALUE TO FDEVICECONTROL IS UNSUPPORTED (FSERR 308):
309 PASSED VALUE TO FDEVICECONTROL IS UNSUPPORTED (FSERR 309):
310 COUNT TO FDEVICECONTROL INSUFFICIENT TO RETURN INFO (FSERR 310):
311 COUNT TO FDEVCNT GREATER THAN AVAILABLE TO STORE INFO (FSERR 311):
312 PASSED SPECIAL CHARACTER HAS PREVIOUSLY DEFINED FUNCTION (FSERR 312):
350 KSAM LOCK TABLE FULL (FSERR 350):
351 TOO MANY RECORD LOCKS IN ONE PROCESS (FSERR 351):
352 RECORD WAS LOCKED ALREADY (FSERR 352):
353 FILE MUST BE UNLOCKED BEFORE ISSUING THIS INTRINSIC (FSERR 353):
354 DATA INTEGRITY VIOLATION (FSERR 354):
355 FILE IS LOCKED BY ANOTHER PROCESS (FSERR 355):
356 LOCK REQUEST REJECTED DUE TO POTENTIAL DEADLOCK (FSERR 356):
357 DATA MAY BE INVALID BECAUSE FILE IS LOCKED BY OTHERS (FSERR 357):

INDEX

-	
date decrement operator	B-11
date subtraction operator	B-11
integer negation operator	B-4
integer subtraction operator	B-4
real subtraction operator	B-13
string removal operator	B-7
time decrement operator	B-14
time subtraction operator	B-14
used to exclude filesets	45
' prefix, used to execute QUERY commands from with MPEX	272
! prefix	
as a prefix character for embedded <, >, >>	29
as a prefix character for variables	33
execute MPEX command instead of a UDC with the same name	96
used as prefix for indirect files	46
used to execute MPEX commands online (default)	19
!'...', MPE syntax for embedded variables	33
![...], MPE syntax for embedded expressions	33
# prefix, used to insert comments into indirect files	47
\$ prefix, used to execute MPEX commands in batch	19
\$CHLOGON-FORBID, STREAMX.DATA.VESoft keyword	141
\$CHLOGON-NOPASS, STREAMX.DATA.VESoft keyword	140
\$CHLOGON-OKMENU, STREAMX.DATA.VESoft keyword	141
\$CHLOGON-PERMIT, STREAMX.DATA.VESoft keyword	141
\$WITHCAPS-FORBID, STREAMX.DATA.VESoft keyword	287
\$WITHCAPS-PERMIT, STREAMX.DATA.VESoft keyword	287
%%, MPEX HOOK command to suspend process	74
%ABORTJOB	
and scheduled jobs, see also %SEC SHOWSCHED	99
and STREAMX scheduled jobs	99
MPEX command	98
see also ACCESSING= User attribute by typing %SEC HELP ACCESSING	98
see also SEC HELP USERSET	98
%ALARM, MPEX command	99
%ALIAS MPEX command	103
%ALLOCATE MPEX command	104
%ALTFILE	
;CREATOR and GROUPID, related POSIX features	111
and non-system volume sets	115
MPEX command	105
%ALTFILE security considerations	106
%ALTJOB, MPEX command	126
%ALTPROC, MPEX command	126
%ALTSCHED, MPEX command	126
%ALTSEC, MPEX command	128

Index

%ALTSPOOLFILE, MPEX command	129
%BACKG, MPEX command	130
%BREAKJOB, MPEX command	135
%BYE, MPEX command	135
%CALC, MPEX command	136
%CALENDAR, MPEX command	137
%CCXL, MPEX command	67
%CHGROUP interaction with \$LOGON-EXECUTE	142
%CHGROUP, MPEX command	137
%CHLOGON	
interaction with \$LOGON-EXECUTE	142
MPEX command	138
%CLEANUP..%ENDCLEANUP, MPEX command	279
%COB74XL, MPEX command	67
%COB85XL, MPEX command	67
%COB85XLK, MPEX command	67
%COBOL, MPEX command	67
%COBOLII, MPEX command	67
%COBOLIIX, MPEX command	67
%COPY using %ALTFILE keywords	144
%COPY, MPEX command	143
%DBADGALT, MPEX command	151
%DBGENALT, MPEX command	151
%DEALLOCATE, MPEX command	151
%DELETEALARM, MPEX command	103
%DELETELVAR, MPEX command	34
%DELETESPOOLFILE, MPEX command	152
%DELETEVAR, MPEX command	153
%DEMO, MPEX command	153
%DEVCONTROL MPEX command	154
%E, synonym for MPEX %EXIT command	159
%ECHO, MPEX command	155
%EDIT, MPEX command	155
%EDITCHG, MPEX command	156
%EDITQUAD, MPEX command	157
%EDITTDP, MPEX command	158
%ELSE, MPEX command	167
%ELSEIF, MPEX command	167
%END, synonym for MPEX %EXIT command	159
%ENDIF, MPEX command	167
%ERASE, MPEX command	158
%ERRCLEAR, MPEX command	159
%ESCAPE, MPEX command	159
%EXIT, MPEX command	159
%FCOPY, MPEX command	159
%FILTER, MPEX command	160
%FORTRAN, MPEX command	67
%FTN, MPEX command	67
%FTNXL, MPEX command	67
%FTNXLLK, MPEX command	67
%GOON, MPEX command	161
%HELP	
for CIERR numbers	162
for FSERR numbers	162
for LOADERR numbers	162

for normal :MPE commands.....	162
for POSIX commands	162
MPEX command	162
See also: %SYNTAX, %SEC SYNTAX, and %VEAUDIT SYNTAX.....	162
using %HELPMAKE to rebuild help files.....	166
%HELPMAKE, MPEX command	165
%HOOK, MPEX command	166
%IDENTIFY	
MPEX command	166
security considerations.....	167
%IF, MPEX command	167
%IFERROR...%ENDIFERROR, MPEX command	279
%INITMPEXMGR	
MPEX command (in CIMGR).....	15
MPEX command (in MPEXMGR).....	11
%INITREDO, MPEX command	11
%INITUDCS, MPEX command	168
%INPUT, MPEX command	169
%KILL, MPEX command to KILL a son process	170
%LISTF	
ID and QEDIT files (normal and 'jumbo')	188
list of modes	172
MPEX command	171
%LISTF mode	
' 0' [FILES].....	173
' 1' [SUMMARY]	174
' 2' [DISC]	174
' 3'	175
' 4'	177
' 5'	178
' 6' [FILENAME] or [QUALIFY].....	179
'-1' [LABEL].....	180
'-2' [ACD].....	180
'-3' [DETAIL] w/lockwords.....	181
ACCESS.....	182
DATES, shows extended date information.....	184
DB, shows IMAGE dataset statistics.....	185
defining your own formats	194
DISCUSE	187
ID, shows extended description	188
POSIX, shows POSIX-related information	192
SAVABLE (by XLTRIM or SQUEEZE).....	189
SEC, shows security information	190
XL3 (MPE/iX :LISTF,3 format)	192
XL4 (MPE/iX :LISTF,4 [security] format)	193
%LISTJOB	
MPEX command	202
see also %SEC SHOWSCHED	202
see also %SHOWJOB.....	264
%MANY, MPEX command	204
%MPEXSTORE, MPEX command	205
%NEWLINK, MPEX command	206
%NOMSG, MPEX command.....	207
%OPTION, MPEX command.....	207
%PASCAL, MPEX command	67

Index

%PASXL, MPEX command	67
%PASXLLK, MPEX command	67
%PAUSE, MPEX command	208
%PAUSEJOB, MPEX command	208
%PRINT, MPEX command	210
%PRINTI, MPEX command	221
%PRINTO, MPEX command.....	221
%PROGINFO, MPEX command.....	222
%PURGE	
Deletes a file or fileset	222
%PURGELINK, MPEX command	225
%QEDIT, MPEX command	226
%QUIT, MPEX command	227
%REDO	
%DO, %LISTREDO abbreviations.....	231
%DO, %LISTREDO in MPEX HOOKed programs	234
%DO, %LISTREDO, MPEX commands.....	227
absolute line numbers	229
editing characters (R,I,D,>,C)	230
finding line to %REDO by contained string.....	229
finding line to %REDO by leading string	229
line ranges	230
redoing the last line.....	229
relative line numbers.....	230
saving command history in a PERMANENT file	233
setting MPEXPROMPT for ease of redo.....	233
%RELEASE, MPEX command	235
%REMOTE HELLO, MPEX command.....	236
%RENAME, MPEX command	237
%REPEAT...%FOREACH, MPEX command	240
%REPEAT...%FORFILES	
MPEX command	240
operating on spool filesets	243
operating on temporary filesets.....	244
%REPEAT...%FORJOBS, MPEX command	245
%REPEAT...%FORNUM, MPEX command	245
%REPEAT...%FORPROFILES, MPEX command (for SECURITY).....	246
%REPEAT...%FORRECS, MPEX command.....	246
%RESUMEJOB, MPEX command	247
%RETURN, MPEX command	247
%RPG, MPEX command	67
%RPGXL, MPEX command	67
%RPGXLLK, MPEX command.....	67
%RUN, MPEX command	248
%RUNACTIVATE, MPEX command	249
%RUNCREATE, MPEX command	249
%RUNINPUT, MPEX command.....	249
%SAVEJOB, %SHOWSAVED, %DOSAVED, MPEX command	250
%SCHEDULE, MPEX command	253
%SEC	
MPEX prefix for SECURITY commands	255
%SEC HELP	
getting help on SECURITY topics from MPEX	164
help on SECURITY	162
%SECURE, MPEX command	255

%SET [NO]CMDTRACECHECK	260
%SET [NO]VARTRACE	261
%SET CAPABILITY	256
%SET CREATORPROTECT	257
%SET DATE	258
%SET DEFAULT	259
%SET GOONMAXPRI	260
%SETLVAR, MPEX command.....	34
%SETVAR, MPEX command.....	262
%SHOW, MPEX command.....	263
%SHOWALARM, MPEX command	102
%SHOWGOON, MPEX command	162
%SHOWJOB, enhanced to show STREAMX scheduled jobs	264
%SHOWJOB, see also %LISTJOB	202
%SHOWLVAR, MPEX command	34
%SHOWME, MPEX command	265
%SHOWOUT, MPEX command.....	266
%SHOWOUTJ, MPEX command	267
%SHOWPAUSED, MPEX command	209
%SHOWPROC, MPEX command	268
%SHOWTREE, MPEX command	270
%SHOWVAR, MPEX command	271
%SPL, MPEX command	67
%SPOONFEED, MPEX command	272
%SPOONINIT, MPEX command	275
%SUBMIT, MPEX command	276
%SYNTAX, MPEX command	278
%TELL, MPEX command	278
%TRAPERROR	
and %ESCAPE	159
MPEX command(s)	279
%UNALIAS, MPEX command.....	104
%UNLESSFAST, MPEX command	14
%VEAUDIT HELP	
getting help on VEAUDIT topics from MPEX	164
help on VEAUDIT	162
%VEAUDIT, MPEX prefix for VEAUDIT commands	282
%VECMDCH, MPEX command	282
%VEOPENCH, MPEX command.....	282
%WARN, MPEX command	283
%WARNF, MPEX command	283
%WHEREIS, MPEX command	284
%WHILE...%ENDWHILE , MPEX command	285
%WITHCAPS, MPEX command.....	286
%XEQ, MPEX command	289
 &	
boolean AND operator	B-5
used to continue commands on multiple lines	97
 *	
integer multiplication operator	B-4
real multiplication operator.....	B-13
string repetition operator	B-7
used as prefix for indirect files	46

Index

, abbreviation for REDO	231
,, abbreviation for LISTREDO.....	231
., Abbreviation for DO.....	231
/	
integer division operator	B-4
real division operator	B-13
when used as the last character of an HFS (POSIX) fileset	52
//	
real division operator	B-13
used to select multiple subdirectories in an HFS (POSIX) fileset	52
:	
prefix, used to indicate an MPE command	17
:ABORT/:RESUME in MPEX (:WAKE)	82
;	
prefix, used to indicate a NOTEPAD command	93
?	
prefix, used to indicate yes/no prompting.....	18
[...], file range wildcard syntax.....	
[STARTBIT:NUMBITS], bit extraction operator	49
[STARTCHAR:NUMCHARS], substring extraction operator	B-6
[STARTCHAR:NUMCHARS], substring extraction operator	B-7
^	
integer exponentiation (raise to power) operator	B-5
real exponentiation (raise to power) operator	B-13
^, used as prefix for indirect files.....	46
boolean OR operator	B-5
+	
date increment operator	B-11
integer addition operator	B-4
real addition operator	B-13
string concatenation operator	B-7
time increment operator	B-14
used to include filesets	45
<	
Command I/O Redirection, input	28
date comparison operator.....	B-11
integer comparison operator.....	B-5
real comparison operator	B-13
string comparison operator	B-9
time comparison operator.....	B-15
<=	
date comparison operator	B-11
integer comparison operator.....	B-5
real comparison operator	B-13
string comparison operator	B-9
time comparison operator.....	B-15
<>	
date comparison operator	B-11
integer comparison operator.....	B-5
real comparison operator	B-13

string comparison operator	B-9
time comparison operator	B-15
=	
date comparison operator	B-11
integer comparison operator	B-5
real comparison operator	B-13
string comparison operator	B-9
target fileset wildcard character	50
time comparison operator	B-15
Used to preserve lockword in target fileset	145
==, target fileset wildcard character	52
>	
Command I/O Redirection, output	28
date comparison operator	B-11
integer comparison operator	B-5
real comparison operator	B-13
string comparison operator	B-9
time comparison operator	B-15
> prefix, used to execute SPOOK5 commands from with MPEX	272
>=	
date comparison operator	B-11
integer comparison operator	B-5
real comparison operator	B-13
string comparison operator	B-9
time comparison operator	B-15
>>	
Command I/O Redirection, append	28
4-digit years in dates	B-2
A few comments about the '!' character	37
ABS, integer absolute value operator	B-4
ABS, real absolute value operator	B-13
ACCDATE, file attribute variable	56
ACCDATETIME, file attribute variable	56
Access date: listing	175
ACCESSIBLE(), file attribute function	59
ACCESSIBLEBY(), file attribute function	59
ACCOUNT	
file attribute	64
file attribute variable	54
job/session attribute variable	65
spool file attribute variable	64
ACCTEXISTS, function	B-18
ACCTIME, file attribute variable	56
ACDREQUIRED, file attribute variable	58
Advanced expression programming	B-41
ALARM, %BACKG task	133
ALARMPRI, BACKG task execution priority variable	133
ALLFILENAMES, %PRINT;SEARCH=	214
ALLTOTALS, %LISTF keyword	172
ALPHA, string test function	B-8
ALPHANUM, string function	B-9

Index

ALREADYSORTED, keyword used in flexible LISTF files	200
Altering ACDs of an entire fileset	128
AND, boolean operator	B-5
ANYPARM and REST\$, command file and UDC parameter qualifier	25
ANYUSER, %SHOWPROC keyword	269
Appending to existing files	150
Appendix	
A: POSIX compatibility issues	A-1
B: VESoft Expressions	B-1
C: Loader Error Messages	C-1
D: Cryptic File System Error Messages	D-1
ASCII, %ALTFIL keyword	125
ASK, %BYE keyword	135
ASL, bit Arithmetic Shift Left operator	B-6
ASR, bit Arithmetic Shift Right operator	B-6
AT=	
%ALARM keyword	99
%PAUSEJOB keyword	209
AUDITC, %BACKG task	133
AUDITCPRI, BACKG task execution priority variable	133
Automatic lockword insertion for %RUN commands	249
BA capability: adding/removing BA from programs	108
BACAP, program file attribute variable	62
BACKGINI.DATA, %BACKG configuration file	133
Background execution of MPEX commands: %GOON	161
BADPASSWORD, function to look up the BADPASS file	B-21
BAND, Bit-wise AND operator	B-6
Batch execution	
of MPEX commands: %SCHEDULE	253
of MPEX commands: %SUBMIT	276
using '\$' Prefix	19
BEGIN, expression program operator	B-42
BESTBLOCKFACTOR, file attribute variable	57
BETWEEN	
date range checking function	B-11
integer range checking function	B-5
real range checking function	B-13
string range checking function	B-9
time range checking function	B-15
used for implementing file ranges	49
BINARY, %ALTFIL keyword	125
BLKFACT, %ALTFIL keyword	106
BLOCKFACTOR, file attribute variable	54
BLOCKSIZE, file attribute variable	54
BNOT, Bit-wise NOT operator	B-6
Boolean	
operators and functions	B-4
variables and constants	B-2
BOR, Bit-wise OR operator	B-6
BOUND, function that checks if variable is defined	B-17
BUF, %PRINT keyword	212
BUFFERSIZE, %COPY keyword	147
BVAR, Boolean VARiable retrieval function	B-17
BVARSET, Boolean VARiable SETting function	B-18

BXOR, Bit-wise eXclusive OR operator	B-6
BYE, %BYE keyword	135
BYTE, %ALTFILE keyword	125
CALENDARTODATE, date function	B-12
CANCELWRITE, expression program operator	B-46
CAP, %ALTFILE keyword	108
CAPABILITY, function	B-16
CASELESS keyword in %PRINT;SEARCH=	217
CCXL	68
CCXLLK	68
CDATE, date formatting function	B-12
CDATEDMY, date formatting function	B-12
CDATEMDY, date formatting function	B-12
CDATEYMD, date formatting function	B-12
CENTER, STRWRITE format specifier	B-38
CHANGE, %SHOWPROC keyword	269
Changing the MPEX prompt	16
CHECKEVERY=	
%ALARM keyword	102
%PAUSEJOB keyword	209
Checking MPEX command success	29
CHLOGON and SECURITY menus	141
CHR, string function (converts integer to ASCII character)	B-8
CI emulation using CI.PUB.VESOFT	15
CI.PUB.SYS	15
CI.PUB.VESOFT (for MPE/iX emulation)	15
CIERR	
%ESCAPE keyword	159
%QUIT keyword	227
CIERROR, JCW set when command fails	29
CIERRORISWARN, JCW set when command fails	29
CIMGR files	15
CL keyword (abbreviation for CASELESS) in %PRINT;SEARCH=	217
CLOCK, time function	B-15
CLOCKTOTIME, time function	B-15
CMDPARM, command file / UDC parameter retrieval function	B-18
CMDPROT, %BACKG task	134
CMDPROTPRI, BACKG task execution priority variable	133
COB74XL	68
COB74XLK	68
COB85XL	68
COB85XLK	68
COBOL	67
COBOLII	67
COBOLIIX	67
CODE, %ALTFILE keyword	110
CODE, file attribute variable	54
Command files	23
Comments	
in expression programs	B-47
in LOGOFF files	B-47
in LOGOFFAB files	B-47
in LOGOFFWR files	B-47
in SECURCON files	B-47

Index

in STREAMX files	B-47
CONTEXT, %PRINT keyword used with ;SEARCH=	215
COPYACCESS, %PRINT keyword	212
COPYACD, %COPY keyword	148
COS, cosine function	B-14
CPULIMIT, job/session attribute variable	66
CREATE, %RENAME keyword	238
Creation date: listing	175
CREATOR	
%ALTFILE keyword	111
file attribute variable	56
Creator ID: listing	175
CREATORACCOUNT, file attribute variable	56
CREDATE, file attribute variable	56
CREDATETIME, file attribute variable	56
CRETIME, file attribute variable	56
CS\$, command file / UDC parameter qualifier	27
CSECURITYA, file attribute variable	58
CSECURITYAC, file attribute variable	58
CSECURITYAL, file attribute variable	58
CSECURITYANY, file attribute variable	58
CSECURITYCR, file attribute variable	58
CSECURITYGL, file attribute variable	58
CSECURITYGU, file attribute variable	58
CSECURITYL, file attribute variable	58
CSECURITYR, file attribute variable	58
CSECURITYW, file attribute variable	58
CSECURITYX, file attribute variable	58
CSL, bit Circular Shift Left operator	B-6
CSR, bit Circular Shift Right operator	B-6
CURRENTCMD, job/session attribute variable	65
D keyword (abbreviation for DELIM) in %PRINT;SEARCH=	218
Data types	B-2
Database I/O functions	B-30
Date	
operators and functions	B-11
variables and constants	B-2
DATE=	
%ALARM keyword	99
%PAUSEJOB keyword	209
DATEADD, date function to increment dates	B-12
DATEBUILD, converts year, month and day into date	B-11
DATEDAY, date function to extract day	B-12
DATEDAYOFWEEK, date function to extract day of week	B-12
DATEMONTH, date function to extract month	B-11
DATEPARSE, date parsing function	B-12
DATETOCALENDAR, date function	B-13
DATEYEAR, date function to extract the 2-digit year	B-11
DATEYEAR4, date function to extract the 4-digit year	B-11
DAY=, %ALARM keyword	99
DAY=, %PAUSEJOB keyword	209
DAYOFYEAR, date function to determine the Julian day	B-12
DBSETBLOCKFACTOR, dataset attribute variable	61
DBSETBLOCKSIZE, dataset attribute variable	62

DBSETBLOCKWASTAGE, dataset attribute variable	62
DBSETCAPACITY, dataset attribute variable	61
DBSETENTRIES, dataset attribute variable	61
DBSETENTRYLENGTH, dataset attribute variable	61
DBSETFULLNESS, dataset attribute variable	61
DBSETHIGHWATERMARK, dataset attribute variable	61
DBSETMAXCAPACITY, dataset attribute variable	61
DBSETMEDIALLENGTH, dataset attribute variable	62
DBSETNAME, dataset attribute variable	61
DBSETNUMPATHS, dataset attribute variable	62
DBSETTYPE, dataset attribute variable	61
DBSIZE, program file attribute variable	63
DBUILD, integer function	B-6
Debugging MPEX command files/UDCs: HPCMDTRACE JCW	15
DELACD	
%ALTFILE keyword	112
%RENAME keyword	238
DELETE, %IDENTIFY keyword	166
DELIM keyword in %PRINT;SEARCH=	218
Determining the status of a VEDB call	B-30
DEV	
%ALTFILE keyword	113
%COPY keyword	147
DEVICE	
file attribute variable	57
spool file attribute variable	64
Device class	
listing	177
of spool files	129
DEVICESECTORS(), file attribute function	59
DIRACCOUNT, account attribute of files	64
DIRGROUP, group attribute of files	64
Disk space	
listing by disk drive	187
listing various information	177
selecting by savable space	57
selecting by used space	54
Disk space, saving	
by lower file limit	118
by making files variable record length	155
by purging unused files	222
by setting max extents=32	116
by SQUEEZEing	124
by XLTRIMing	124
Displaying the system name in %LISTF headings	172
DL, program file attribute variable	63
DMY, READDATE format specifier	B-36
DOWNSHIFT	
READxxx format specifier	B-36
STRWRITE format specifier	B-38
DS capability: adding/removing DS from programs	108
DSCAP, program file attribute variable	62
DWNS, string downshift function	B-8
ELSE, expression program operator	B-41

Index

Embedded quote characters (") in string variables	B-2
END, %PRINT keyword	216
END, expression program operator	B-42
ENTRY=, specifying entrypoint for implied RUN	23
EOF, file attribute variable	54
Eugene Volokh	i
Examples	
Database Management	5
File System Security	6
Job Stream Programming	6
Managing Disk space	3
MPEX commands	2
Program Development	4
System Management	2
EXCLUSIVE, file attribute variable	58
EXP, exponentiation ('e' to the R) function	B-14
Expression program variables	B-44
Extent map: listing	177
EXTENTS, %ALTFIL keyword	116
EXTENTSIZE, file attribute variable	57
FALSE, boolean constant	B-5
FANYEXISTS, function that checks if temp OR perm file exists	B-22
FANYINFO, retrieves information on temp OR permanent files	B-23
FCONTAINS, file attribute function (string search)	60
FEXISTS, function that checks if a file exists	B-22
File	
attribute functions	B-21
attributes: variables and functions	52
I/O functions	B-26
ranges	48
FILE	
file attribute variable	54
spool file attribute variable	64
Filecode	
171: expression programs	B-47
176: compiled expression programs	B-47
FILEGROUP, file attribute variable	56
FILES=, %SHOWPROC keyword	270
Fileset wildcards: ?, @, #, [...]	48
Filesets	
Enhanced POSIX Filenames	42
Enhanced POSIX Filesets	43
General MPE and POSIX filesets	43
Standard MPE filenames	41
Standard MPE filesets	41
FINALLY, final processing in flexible LISTF files	198
Finding files with non-existent creators	113
FINFO, function to retrieve file information	B-23
FIRST, initial processing in flexible LISTF files	198
FIXED, %ALTFIL keyword	125
FLABLDEV, file attribute variable	57
Flexible %LISTF	194
FLIMIT	
%ALTFIL keyword	118

setting it to EOF	124
FLIMIT, file attribute variable	54
FMTCAPS, program file attribute	62
FMTFOPTIONS, file attribute variable	56
FMTJOB, job/session attribute variable	65
FMTLOGON, job/session attribute variable	65
FMTOPENED,	
file attribute variable	54
FMTRECSIZE, file attribute variable	54
FMTTYPE, file attribute variable	54
FOPTIONS, file attribute variable	56
FOR RFILE IN, fileset-handling in flexible LISTF files	196
FOR, expression program operator	B-43
FORMAT	
%PRINT keyword to format output lines	213
=FILES, %SHOWPROC keyword to show files open by a process	269
FORTRAN	67
FPERMEXISTS, function that checks if a permanent file exists	B-22
FPERMINFO, function to retrieve permanent file information	B-23
FREMLockWORD, filename lockword removal function	B-21
FSEARCHEXP, file attribute function (string search)	60
FSEARCHSTRING, file attribute function (string search)	60
FSERROR, JCW set when command fails	29
FTEMP EXISTS, function that checks if a temp file exists	B-22
FTEMPINFO, function to retrieve temp file information	B-23
FTN	67
FTNXL	68
FTNXLLK	68
FULLNAME, file attribute variable	58
FULLNAMELOCK, file attribute variable	58
GARBAGE, STRWRITE format specifier	B-38
GOD -- gives SM capability, :ALLOWs all commands	89
GOD, see also SEC HELP ALLOW (\$ALLOW in SECURCON)	89
GOON, %RUN keyword	84
GROUP	
file attribute	64
file attribute variable	54
GROUP, job/session attribute variable	65
GROUPEXISTS, function	B-18
HASACD, file attribute variable	58
HASFORMSMMSG, spool file attribute variable	64
HASPRIVSEG, program file attribute variable	62
HELLO, %BACKG task	134
HELLOPRI, BACKG task execution priority variable	133
HEX, READINTEGER format specifier	B-36
HEX, string function (converts integer to hexadecimal)	B-8
HEX, STRWRITE format specifier	B-38
HFS filesets	
and the '-' character	51
and the '/' character	51
and the '@' character	51
and the '=' character	51
with '==' (for TARGET filesets)	52

Index

HHMMSS, time function	B-15
HIGHLIGHT, %PRINT keyword	213
HIGHORDER, integer function	B-6
Hints, how to disable them	11
HOMEGROUP, job/session attribute variable	65
HOOKing programs	77
HPAUTOCONTTOPELVEL, MPEX control variable	9
HPDTCPORTID, job/session attribute variable	66
HPPATH, MPEX variable indicating command search path	22
HPREDOFLIST, MPEX control JCW	235
HPREDONODUPS, MPEX control JCW	235
HPREDOSIZE, MPEX control JCW/variable	234
HPREDOVEMODIFY, MPEX control JCW	235
HPSYSNAME, variable used in %LISTF headings	39
HPTERMINAL(), function that checks for HP terminals	B-20
I2, STRWRITE format specifier	B-38
I4, STRWRITE format specifier	B-38
IA capability: adding/removing IA from programs	108
IACAP, program file attribute variable	62
ICODE, file code parsing function	B-16
ID, file attribute variable	58
IDCONTAINS(S) example	189
File attribute function	58
IF, expression program operator	B-41
IFLOW, %PRINTO keyword	221
IMAGE databases copying	146
moving from disk to disk	113
releasing	235
renaming	237
securing	255
Implied RUN	21
Important security considerations for the MPEXMGR file	12
IN= %ALARM keyword	99
%PAUSEJOB keyword	209
Indirect Files	46
INFO= with :RUN MAIN.PUB.VESOFIT	10
with implied RUN	23
INITEXTENTS, %ALTFILE keyword	121
INMSG, %RUN keyword	82
INPRI, job/session attribute variable	65
INPUT %RUN command parameter	81
function, accepts input from user	B-16
see also %RUNCREATE / %RUNINPUT / %RUNACTIVATE	249
INSIDEMPEX, JCW to determine whether or not you're in MPEX	27
INSPOOLFILENUM, job/session attribute variable	66
INTCODE, file attribute variable	54
Integer operators and functions	B-4
variables and constants	B-2

INTEGER	
byte-to-integer conversion function.....	B-6
real-to-integer truncation function	B-14
INTEGERPARSE, integer parsing function.....	B-6
INTRODATE, job/session attribute variable	65
Introduction.....	1
INTROTIME, job/session attribute variable.....	65
ISACTIVE, spool file attribute variable	63
ISASCII, file attribute variable	54
ISBINARY, file attribute variable	54
ISBOSS, file attribute variable	58
ISBYTESTREAM, file attribute variable.....	55
ISCCTL, file attribute variable	54
ISCHGROUPED(), function to test %CHGROUP status	138
ISCHLOGONED()	
differences from ISCHGROUPED()	140
function that checks if a user has done :CHLOGON to another logon	B-17
function to test %CHLOGON status	140
ISCIR, file attribute variable.....	55
ISCMPROG, file attribute variable	55
ISDEFERRED	
job/session attribute variable.....	66
Spool file attribute variable	63
ISDIR, file attribute variable.....	55
ISENCRYPTED, file attribute variable	55
ISEXEC, job/session attribute variable.....	66
ISEXECUTABLE, file attribute variable.....	55
ISFIXED, file attribute variable	54
ISHIDDEN	
example showing usage of ISHIDDEN file attribute.....	44
file attribute variable	55
ISINIT, job/session attribute variable	66
ISJUMBO, file attribute variable	55
ISKSAM, file attribute variable.....	55
ISKSAM64, file attribute variable.....	55
ISKSAMXL, file attribute variable	55
ISLINK, file attribute variable	55
ISLOCKED, spool file attribute variable	63
ISMSG, file attribute variable.....	55
ISNMPROG, file attribute variable	55
ISNOCCTL, file attribute variable	54
ISOCTCOMPED, program file attribute variable.....	63
ISOPENED, spool file attribute variable	63
ISPRIV, file attribute variable.....	55
ISPROG, file attribute variable	55
ISQEDIT, file attribute variable	55
ISQUIET, job/session attribute variable	66
ISREADY, spool file attribute variable.....	63
ISRELEASED, file attribute	57
ISRESTARTABLE, job/session attribute variable.....	66
ISRIO, file attribute variable	55
ISSCHED, job/session attribute variable	66
ISSECURED, file attribute variable.....	57
ISSPSAVE, spool file attribute variable.....	63
ISSTD, file attribute variable	54

Index

ISSTDLISTDELETE, job/session attribute variable	66
ISSUSP, job/session attribute variable	66
ISUNDEFINED, file attribute variable	54
ISVARIABLE, file attribute variable	54
ISWAIT, job/session attribute variable	66
ISXLSPOOLFILE, file attribute variable	55
IVAR, Integer VARiable retrieval function	B-17
IVARSET, Integer VARiable SETting function	B-18
JOBABORTED, spool file attribute variable	64
JOBNUMBER, spool file attribute variable	64
JOBTYPE, spool file attribute variable	64
JSCOUNT, function to count jobs or sessions in userset	B-19
JSNAME	
job/session attribute variable	65
spool file attribute variable	64
KEEPALLOW, %CHLOGON keyword	140
KEEPAMDATES	
%ALTFILE keyword	121
%COPY keyword	148
%PRINT keyword	213
%RELEASE keyword	236
%RENAME keyword	238
%SECURE keyword	255
KEEPATTR, %COPY keyword	144
KEEPCAPS, %CHLOGON keyword	140
KEEPTRAIL, %ALTFILE keyword	125
KEEPUDCS, %CHLOGON keyword	140
KEYFILE, %COPY keyword	146
KILL, %RUN keyword	85
LABELEOF, file attribute variable	58
LABELFLIMIT, file attribute variable	58
LASTTEXTENTSIZE, file attribute variable	57
LDELIM keyword in %PRINT;SEARCH=	218
LDEVIN, job/session attribute variable	65
LDEVLIST, job/session attribute variable	65
LEFTJUST	
integer left-justification	B-38
string left-justification	B-37
LEN, string length function	B-7
LFT, string function	B-8
LINENUM, %PRINT;SEARCH= current line number variable	220
List of MPE FINFO parameters, by name and number	B-23
LOADED, file attribute variable	58
Local variables	34
LOCALGROUPID	
%ALTFILE keyword	111
%RENAME keyword	238
LOCATTR, job/session attribute variable	66
Lockword	
on the %COPY command	149
on the %RENAME command	239
LOCKWORD	

%ALTFILE keyword	122
file attribute variable	56
Lockwords	
preserving them on %COPY/%FCOPY/%RENAME	89
LOG, logarithm function	B-14
LOGOFF, %BACKG task	134
LOGOFFPRI, BACKG task execution priority variable	133
LOGON	
%ALARM command keyword	99
%BACKG task	134
LOGONPRI, BACKG task execution priority variable	133
LOWORDER, integer function	B-6
LSL, bit Logical Shift Left operator	B-6
LSR, bit Logical Shift Right operator	B-6
LTRIM, string function (strips leading char)	B-7
MAINPIN, job/session attribute variable	66
MATCHES(), file attribute function	60
MATCHES, string pattern matching operator	B-9
MAX	
%PRINT;SEARCH= keyword	216
date maximum function	B-11
integer maximum function	B-5
real maximum function	B-13
string maximum function	B-9
time maximum function	B-15
MAXCONSECUTIVE, string function	B-9
MAXDATA	
%ALTFILE keyword	123
program file attribute variable	63
MAXEXTENTS, file attribute variable	54
MDY, READDATE format specifier	B-36
MIN	
date minimum function	B-11
integer minimum function	B-5
real minimum function	B-13
string minimum function	B-9
time minimum function	B-15
MOD, modulo (remainder) operator	B-4
MODDATE, file attribute variable	56
MODDATETIME, file attribute variable	57
Modification date, listing	175
Modification time, listing	175
MODTIME, file attribute variable	56
MORTAL -- undoes a previous 'GOD' operation	90
MOVING GROUPS TO A NON-SYSTEM VOLUME SET	116
MPE	
command execution function	B-16
commands	17
prefix, used to indicate an MPE command	17
MPE/iX compilers	67
(w/linking)	67
MPE/V and MPE/iX differences	
%PROGINFO command	222
compiled exp progs	B-47

Index

MPEX	
and your system security.....	86
batch job aborts	9
commands	96
process handling.....	78
variables: general discussion	32
MPEX and MPE/iX.....	16
MPEX HOOK.....	71
preserves ACDs in HOOKed programs.....	76
MPEXALARMPREFIX, variable to prefix to %ALARMS	40, 101
MPEXALTFILEKEEPAMDATES, MPEX JCW	122
MPEXCMDTRACE JCW	38
MPEXCONTINUECOLON, MPEX control JCW	10
MPEXCOPYKEEPAMDATES, MPEX JCW	148
MPEXCRTRESTRICTxxx, MPEX variable	31
MPEXCURRENTFILE	
current file variable in %REPEAT...%FORFILES.....	241
current spoolfile variable	244
MPEXDBADGFILE, variable indicating where ADAGER is	39
MPEXDBGENFILE, variable indicating where DBGGENERAL is	39
MPEXDEFAULTJOBNAME, MPEX variable for \$ commands	19
MPEXDEFAULTJOBPARMS, MPEX variable	19
MPEXFASTSTART, MPEX JCW to set 'faststart' mode	13
MPEXFILESET, keyword used in flexible LISTF files.....	200
MPEXFILESETHD, keyword used in flexible LISTF files.....	197
MPEXFILESETTOTAL, totaling construct used in flexible LISTF files.....	196
MPEXFINFOANY, MPEX JCW	B-21
MPEXHEADER1, header variable used in flexible LISTF files.....	196
MPEXHEADER2, header variable used in flexible LISTF files.....	196
MPEXHEADER3, header variable used in flexible LISTF files.....	196
MPEXLISTFACCESSCHUNKSIZE, MPEX JCW	184
MPEXLISTFDBMODE5, MPEX JCW	187
MPEXLISTFLABELSIZE, variable used to control LISTF,-1 display of file labels.....	A-3
MPEXLISTFNODAYS, variable used to control LISTF,2 display of file age	A-3
MPEXMGR files	10
MPEXNUMFAILED, JCW set by MPEX	30
MPEXNUMSUCCEEDED, JCW set by MPEX	30
MPEXPIN, JCW set by MPEX	86
MPEXPREFIX=, used to specify alternate trigger character for %HOOK.....	166
MPEXPREFIXDEFnn	
MPEX variable used for SPOONFEEDing.....	274
variables used (mostly) for SPOONFEEDing	40
MPEXPREFIXNOTE, variable configuring NOTEPAD prefix character	95
MPEXPRINTKEEPAMDATES, MPEX JCW.....	213
MPEXPRINTLINESFOUND, MPEX variable.....	220
MPEXPRINTNONDELIMS, variable	218
MPEXPROMPT, variable indicating MPEX's prompt.....	39
MPEXQEDITFILE, variable indicating where QEDIT is.....	39
MPEXREDOSIZE	
MPEX control JCW/variable	234
variable used to replace HPREDOSIZE	A-7
MPEXSONEXECPIN, JCW of a %SPOONFED process	275
MPEXSTREAMGOD, MPEX JCW	20
MPEXSTREAMNOVERIFY, MPEX JCW.....	20
MPEXTELLBACK, variable indicating whom MPEX jobs :TELL to	39

MPEXTELLBACKCMD, variable indicating how MPEX jobs notify	40
MPEXWARNFPREFIX	
example of use	283
variable to prefix %WARNF messages	40
MR capability: adding/removing MR from programs	108
MRCAP, program file attribute variable	62
MUSTEXEC, %DOSAVED keyword	252
NETPUSH, %BACKG task	134
NETPUSHPRI, BACKG task execution priority variable	133
NETPUSHTIME, variable set in BACKGINI.DATA.VESOFT	134
NETRECV, %BACKG task	134
NETRECVPRI, BACKG task execution priority variable	133
NEW, %RUN keyword	85
NEWERTHAN(), file attribute function	60
NMATCHES, string pattern matching operator	B-9
NMHEAP, program file attribute variable	63
NMSTACK, program file attribute variable	63
NOACTIVATE, %RUN keyword	85
NOACTSONTERM, %RUN keyword	248
NOCHECK	
%ALTFILE keyword	111
%IDENTIFY keyword	166
NOCR, READxxx format specifier	B-36
NOECHO, READxxx format specifier	B-36
NOEXPANDDIR, %LISTF keyword	171
NOFILEPAGEBREAK, %PRINT keyword	213
NOHOLIDAY, %CALENDAR keyword	137
NOHP, %ALARM command keyword	100
NONUMRECOGNIZE, %PRINT keyword	213
NOPAGEHEAD, %LISTF keyword	172
NORMTOTALS, %LISTF keyword	172
NOT, boolean operator	B-5
NOTEPADS	93
configure filename using VESOFTNOTEFILE variable	95
editing with [:M]odify and [:D]elete	94
Entering notes using ; prefix character	93
Printing entries	94
Searching for entries	94
using multiple NOTEPAD files	95
NOTOTALS, %LISTF keyword	172
NOTREE, %SHOWPROC keyword	269
NOVERIFY, %PAUSEJOB keyword	209
NTOKEN, string parsing function	B-10
NUM, %PRINT keyword	214
Number of copies of spool files	129
NUMCOPIES	
job/session attribute variable	65
spool file attribute variable	64
NUMERIC, string function	B-8
NUMEXTENTS, file attribute variable	54
NUMLINES, spool file attribute variable	64
NUMSEGS, program file attribute variable	62
OBJECTFILE()	

Index

fileset processing function	242
function for use with %REPEAT...%FORFILES	242
OBJECTFILE, 'to-file' name function	B-21
OBSFILL, %BACKG task	135
OBSFILLPRI, BACKG task execution priority variable	133
OBSFILLTIME, variable set in BACKGINI.DATA.VESOFT	135
OCTAL	
READINTEGER format specifier	B-36
string function (converts integer to OCTAL string)	B-8
STRWRITE format specifier	B-38
ODD, integer testing function	B-5
OFF, boolean constant	B-5
OLD, %PRINT keyword	214
OLDANY, %PRINT keyword	214
OLDTEMP, %PRINT keyword	214
ON, boolean constant	B-5
ONBREAKAFTER, level-breaking in flexible LISTF files	198
ONBREAKBEFORE, level-breaking in flexible LISTF files	198
ONDEVICE(), file attribute function	60
OPENED, file attribute variable	54
OPENEDSYSFAIL, file attribute variable	59
Opening a database for access	B-30
Operator precedence	B-1
OR, boolean operator	B-5
ORD, string function (returns ASCII value of character)	B-8
Other variable features	35
OUT, %PRINT keyword	214
OUTPRI	
job/session attribute variable	65
spool file attribute variable	63
Output formatting functions	B-37
Output priority of spool files	129
OUTSPOOLFILENUM, job/session attribute variable	66
OWNER, file attribute variable	56
PAGE, %PRINT keyword	214
PAGEHEAD	
%PRINT keyword	214
page break processing in flexible LISTF files	198
PARM	
PARM=1, MPEX's 'faststart' mode	13
specifying PARM= with implied RUN	23
used in header record for command files	24
PASCAL	67
PASXL	68
PASXLLK	68
PATH, file attribute variable	59
PAUSE, function	B-18
PH capability: adding/removing PH from programs	108
PHCAP, program file attribute variable	62
PMCAP, program file attribute variable	62
POS, string function (finds one string in another)	B-7
PREV=, %PRINT keyword, used with ;SEARCH=	215
PRI=, %RUN keyword	84
Printing conventions	xi

PRINTOPREPLY	
console reply request function	B-16
console reply request function-example	7
PRIVATE, job/session attribute variable	66
Privileged Mode	
adding/removing PM from programs	108
finding PM programs	62
finding programs with PM segments	62
QUIET, %PAUSEJOB keyword	209
QUIT, %BYE keyword	135
R	
current record variable within %PRINT;FORMAT= expression	213
current record variable within %PRINT;SEARCH= expression	219
RANDOM, integer function	B-5
RANDOMNAME, function	B-18
RDELIM keyword in %PRINT;SEARCH=	218
READDATE, date input function	B-35
READING, file attribute variable	59
READINTEGER, integer input function	B-35
READREAL, real input function	B-35
READSELECT, input function	B-35
READSTRING, string input function	B-35
READTIME, time input function	B-35
READYDATE, spool file attribute variable	64
READYTIME, spool file attribute var	64
Real	
operators and functions	B-13
variables and constants	B-3
REAL, integer-to-real conversion function	B-14
REALPARSE, real number parsing function	B-14
REC, %ALTFILE keyword	125
RECNUM, %PRINT;SEARCH= current record number variable	219
RECSIZE, file attribute variable	54
Redirecting	
command input/output	28
errors and warnings during command execution	A-4
Released files: finding them	57
REMTOKEN, string parsing function	B-10
REPETITIVE ALARMS	101
Restore date: listing	175
RESTORED, file attribute variable	59
Restricting MPEX commands to only work on certain files	31
RFILE., prefix for MPEX file attribute variables	53
RHT, string function	B-8
RIGHTJUST, STRWRITE format specifier	B-38
RJOB	
prefix for Job/Session attributes	65
used in %REPEAT...%FORJOBS commands	245
ROUND, real-to-integer conversion function	B-13
RPG	67
RPGXL	68
RPGXLLK	68
RPT, string function	B-8

Index

RSTDATE, file attribute variable	57
RSTDATETIME, file attribute variable	57
RSTTIME, file attribute variable	57
RTRIM	
READxxx format specifier	B-36
string function (strips trailing char)	B-7
Running MPEX	8
in batch	9
interactively	8
with ;INFO	10
SAVABLESECTORS, file attribute variable	57
SCODE, file code formatting function	B-16
SEARCH, %PRINT keyword to find strings	214
SEARCH=IFTRUE or SEARCH=IFFALSE, %PRINTO keyword	221
Searching for	
delimited strings (strings as words)	218
lines by line number	220
lines containing all of a list of strings	217
lines containing any of a list of strings	217
patterns in fileset records using wildcards	219
strings ignoring case	217
SECTORS, file attribute variable	54
Security	
%ALTSEC	86
%COPY	87
%RELEASE	86
%RENAME	86
%SECURE	86
copying files into other accounts	87
databases	87
IMAGE	87
Lockwords	88
removing creator-only restrictions	86
spool files	87
SECURITY user profile attributes	66
SECURITYA, file attribute variable	58
SECURITYAC, file attribute variable	58
SECURITYAL, file attribute variable	58
SECURITYCR, file attribute variable	58
SECURITYGL, file attribute variable	58
SECURITYGU, file attribute variable	58
SECURITYL, file attribute variable	58
SECURITYLOG(..), function to write a SECURITY log record	B-21
SECURITYR, file attribute variable	58
SECURITY-related functions	B-20
SECURITYW, file attribute variable	58
SECURITYX, file attribute variable	58
SELECT, keyword used in flexible LISTF files	197
Selection criteria	
for files	52
for files (by group/account attribute)	64
for normal files	53
introduction	47
SECURITY user profile attribute	66

Self demo	1
SESSION, %ALARM command keyword	99
Setting	
default parms for MPEX commands.....	30
execution priority for BACKG tasks	133
MPEX variables	34
SETVAR, arbitrary VARiable SETting function	B-18
SHOW	
%BACKG keyword	132
SHOW,taskname %BACKG keyword.....	132
SILENT, %CHLOGON keyword	140
SIN, sine function	B-14
SONALIVE(I), son process checking function	B-17
SONALIVENAME(I), son process checking function	B-17
SORT, keyword used in flexible LISTF files	201
SOURCE and TARGET filesets using HFS syntax	51
Special MPE access functions	B-16
Special variables used by MPEX.....	38
SPECIFIC DIFFERENCES BETWEEN MPE AND MPEX COMPILE COMMANDS.....	69
Specifying commands to be executed when MPEX is entered	10
Speeding up MPEX start-up	13
SPL	67
Spool files	
deleting by output priority	152
selecting by creating job/session name	266
selecting spool files of aborted jobs.....	266
SPOOL., prefix for MPEX spoolfile attribute variables	244
SPOOL.attrvar, spoolfile attribute variables	63
SPOOLFILENUM, spool file attribute variable	64
SPOONFEEDing, passing input to son processes	273
SPSAVE, job/session attribute variable	66
SQRT, square root function	B-14
SQUEEZE, %ALTFILe keyword	124
STACK	
%ALTFILe keyword	124
program file attribute variable	63
Stack overflows	
in %LISTF,ACCESS, how to avoid them	184
in %PRINT, how to avoid them	212
START	
%BACKG keyword	132
%PRINT keyword	216
STARTJOB, %BACKG keyword	131
STATE, job/session attribute variable.....	66
STATEDATE, file attribute variable	57
STATEDATETIME, file attribute variable	57
STATETIME, file attribute variable	57
STDIN=, %RUN key.....	84
STDLIST=, %RUN keyword	84
STOP, %BACKG keyword.....	132
STOPJOB, %BACKG keyword.....	131
STORED, file attribute variable.....	59
STR, substring extraction function	B-8
STRCHANGE, string function	B-8
STRCOUNT, string function	B-8

Index

String operators and functions	B-7
String variables and constants	B-2
STRLDROP, string function (drop leading characters)	B-7
STRLTRIM, string function (strips leading char)	B-7
STRRDROP, string function (drop trailing characters)	B-7
STRRTRIM, string function (strips trailing char)	B-7
STRWRITE	
date formatting function	B-39
integer formatting function	B-38
real formatting function	B-40
string formatting function	B-37
time formatting function	B-40
SVAR, String VARIABLE retrieval function	B-17
SVARSET, String VARIABLE SETting function	B-18
SYS, %IDENTIFY keyword	166
TAN, tangent function	B-14
Target filesets	50
TARGET filesets, when the target is a directory	52
TASKLIST.BACKG.VESOF, BACKG configuration file	133
Technical Support	i
TERMCLEAR(), function that clears screen on HP terminals	B-20
TERMFKEY(), function that returns HP function key sequence	B-20
TERMFKEYNUMBER(S), function that returns HP function key number	B-20
Terminal input functions	B-35
VEMODIFY	B-36
Terminal manipulation functions	B-19
TERMINAL, %ALARM command keyword	99
TERMPOS(I1,I2), function to position cursor on HP terminals	B-20
THEN, expression program operator	B-41
Time	
constants	B-14
operators and functions	B-14
variables and constants	B-3
TIMEBUILD, time function	B-15
TIMEPARSE, time function	B-15
TO=userset, %ALARM command keyword	99
TODAY, date function	B-12
TOKEN, string parsing function	B-10
TOTAL, function used in flexible LISTF files	199
Trapping program output (\$STDLIST)	B-34
TREE, %SHOWPROC keyword	269
TRUE, boolean constant	B-5
TRUNC, %SHOWPROC keyword	269
TRY ... CLEANUP, expression program operator	B-44
TRY ... RECOVER, expression program operator	B-44
TYPEOF, function that returns the type of an expression	B-17
UNDEF, %ALTFIL keyword	125
UNN, %PRINT keyword	213
UNSIGNED, STRWRITE format specifier	B-38
UPS, string upshift function	B-8
UPSHIFT	
READxxx format specifier	B-36
STRWRITE format specifier	B-38

USER	
%SHOWPROC keyword.....	269
job/session attribute variable.....	65
pool file attribute variable	64
USEREXISTS, function.....	B-18
Using	
;WAIT, ;IFLOW and ;SEARCH= to view \$STDLISTs.....	221
abbreviated logons with %CHLOGON	142
MPEX to compile programs	67
MPEX variables	32
PRINT;OUT= without issuing a file equation	211
third party compilers with MPEX.....	70
VALIDCOLDLOADID, file attribute variable.....	58
VALIDDATE, date checking function.....	B-12
VALIDINTEGER, integer checking function	B-6
VALIDTIME, time function.....	B-15
VAR, creating expression program variables.....	B-44
VARIABLE, %ALTFILE keyword	125
Variables	B-1
VEACCTINFO, function	B-19
VEAUDITCTIME, variable set in BACKGINI.DATA.VESOF.....	133
VECURRFILENAME, variable set to the current command filename	40
VEDBCHAINLEN, predefined IMAGE variable	B-30
VEDBCLOSE, database I/O function	B-33
VEDBCONTROL, database I/O function.....	B-33
VEDBDATALEN, predefined IMAGE variable.....	B-30
VEDBDELETE, database I/O function	B-32
VEDBDELETEK, database I/O function	B-32
VEDBERR, variable set by VEDBxxx functions.....	B-30
VEDBFIND, database I/O function	B-31
VEDBGET, database I/O function	B-31
VEDBGETV, database I/O function.....	B-32
VEDBLOCK, database I/O function.....	B-32
VEDBNEXTREC, predefined IMAGE variable	B-30
VEDBOPEN, database I/O function	B-31
VEDBPREVREC, predefined IMAGE variable	B-30
VEDBPUT, database I/O function	B-32
VEDBPUTL, database I/O function.....	B-32
VEDBRECNUM, predefined IMAGE variable	B-30
VEDBUNLOCK, database I/O function.....	B-33
VEDBUPDATE, database I/O function	B-32
VEDBUPDATEK, database I/O function	B-32
VEEOF	
variable set by certain I/O functions	B-28
variable set by VEDBGET	B-31
variable set by VEDBGETV	B-32
variable set by VEFINDBYKEY	B-28
variable set by VEFREAD	B-27
variable set by VEFREADBYKEY	B-28
variable set by VEFREADC.....	B-28
variable set by VEFREADCV	B-28
variable set by VEFREADV.....	B-27
VEFANYINFO, retrieves information on temp OR permanent files	B-22
VEFCLOSE, file I/O function.....	B-26

Index

VEFCONTROL, file I/O function	B-27
VEFFINDBYKEY, file I/O function	B-28
VEFFINDN, file I/O function	B-28
VEFINFO	
function to retrieve file information.....	B-22
open file info function	B-27
VEFLOCK, file I/O function	B-27
VEFOPEN, file I/O function	B-26
VEFPERMINFO, function to retrieve permanent file information.....	B-22
VEFPOINT, file I/O function	B-27
VEFREAD, file I/O function.....	B-27
VEFREADBYKEY, file I/O function	B-28
VEFREADBYKEYV, file I/O function	B-28
VEFREADC, file I/O function.....	B-28
VEFREADCV, file I/O function.....	B-28
VEFREADDIR, file I/O function	B-27
VEFREADLABEL, file I/O function	B-27
VEFREADV, file I/O function.....	B-27
VEFREMOVE, file I/O function	B-28
VEFSERR, variable set by certain I/O functions	B-28
VEFTEMPINFO, function to retrieve temporary file information.....	B-22
VEFUNLOCK, file I/O function	B-27
VEFUPDATE, file I/O function.....	B-28
VEFWRITE, file I/O function	B-27
VEFWRTEDIR, file I/O function	B-27
VEFWRITELABEL, file I/O function	B-27
VEFxxx functions, file access for SM users.....	B-26
VEGROUPINFO, function.....	B-19
VEJOBINFO	
attributes.....	65
function	B-19
VEMODIFY.....	91
configure operation using VESOFTVEMODIFYEXPANDOK.....	A-1
Control-^ (insert -- same as Control-B).....	93
Control-A (Append)	91
Control-B (insert Before)	91
Control-C (change Case)	92
Control-D (Delete character)	92
Control-E (Erase rest of line)	92
Control-F (Find character)	92
Control-G ('Goof': undo all modifications)	92
Control-L (Lengthen -- same as Control-A)	92
Control-N (display line or command Number)	92
Control-O (enter Overwrite mode).....	92
Control-Q ('Question': help)	92
Control-S (Scan for character backwards)	92
Control-T (Terminate mode)	92
Control-X ('cancel': redisplay line).....	93
Extended to multi-line commands	A-1
line editing made easier	90
VEMODIFY()	
callable function	91
example of use	91
VEPARMS, expression program parameters.....	B-48
VEPRINTFINFO, file I/O function.....	B-27

VEPROC.JOB.VESOFT	78
VEPROFILEEXISTS, function.....	B-20
VEPROFILEINFO, function	B-20
VERIFY, %PAUSEJOB keyword.....	209
VESOFT address, phone, FAX	i
VESOFTALARMPAUSE, variable set in BACKGINI.DATA	133
VESOFTALARMPAUSE, variable used by %ALARM command	102
VESOFTCONTINUENOSPACE, JCW to control line continuation	97
VESOFTDEFAULTALARM, MPEX variable	30
VESOFTDEFAULTALTFILE, MPEX variable.....	30
VESOFTDEFAULTALTJOB, MPEX variable	30
VESOFTDEFAULTBREAKJOB, MPEX variable	30
VESOFTDEFAULTBYE	
forcing %BYE to %QUIT without logging off	136
forcing %BYE to prompt whether to logoff	136
MPEX variable.....	30
VESOFTDEFAULTCHLOGON, MPEX variable	30
VESOFTDEFAULTCOPY, MPEX variable	30
VESOFTDEFAULTDEVCONTROL, MPEX variable	30
VESOFTDEFAULTDOSAVED, MPEX variable	30
VESOFTDEFAULTIDENTIFY, MPEX variable	30
VESOFTDEFAULTIMPRUN, MPEX variable	30
VESOFTDEFAULTPAUSEJOB, MPEX variable	30
VESOFTDEFAULTPREP, MPEX variable.....	30
VESOFTDEFAULTPREPRUN, MPEX variable	30
VESOFTDEFAULTTPRINT, MPEX variable	30
VESOFTDEFAULTRENAME, MPEX variable.....	30
VESOFTDEFAULTRESUMEJOB, MPEX variable	30
VESOFTDEFAULTRUN, MPEX variable.....	30
VESOFTDEFAULTSAVEJOB, MPEX variable	30
VESOFTDEFAULTSHOWALARM, MPEX variable.....	30
VESOFTDEFAULTSHOWJOB, MPEX variable	30
VESOFTDEFAULTSHOWME, MPEX variable	30
VESOFTDEFAULTSHOWPROC	
MPEX variable.....	30
overriding ;TRUNC with ;NOTRUNC	269
VESOFTDEFAULTxxx, variables to set defaults for MPEX commands	40
VESOFTGOONPRI, variable to control %GOON queue	162
VESOFTHOOKKNOKEEPACD, HOOK JCW to avoid ACD keep.....	77
VESOFTHOOKPPSONONLY, HOOK JCW to forbid %% from CI:RUN programs	75
VESOFTNOTALWAYSPOSIX, variable used to control LISTF display of POSIX files	A-2
VESOFTNOTEFILE, configuring name of NOTEPAD file	95
VESOFTPAGESIZE, JCW used to set %LISTF page size	172
VESOFTSORTFILESIZE	
JCW that sets sort file size for %LISTF	172
JCW to set flexible %LISTF sort file size	202
VESOFTVEMODIFYBIT8CHECK, MPEX control JCW	235
VESOFTVEMODIFYEXPANDOK	
JCW to control display	235
variable used to control VEMODIFY operation	A-1
VEUSERINFO, function	B-19
Visual %REDO extends beyond a single screen line	90
WAIT	
%PRINT keyword	216

Index

%PRINTO keyword.....	221
WAKEUP.PUB.VESOFT	82
WHILE, expression program operator	B-42
WITHCAPS, security restrictions and other notes	288
WRITE, expression program operator	B-46
Writeln, expression program operator	B-46
WRITEPEMESSAGE, writes MPE error message	B-16
WRITEPEXLSTATUS, formats MPE/iX status (error) code.....	B-16
WRITEPAGE, expression program operator	B-46
WRITEPROMPT, expression program operator	B-46
Writing expression programs	B-47
WRITING, file attribute variable.....	59
XLTRIM, %ALTFILE keyword.....	124
XOR, boolean eXclusive OR operator.....	B-5
Yes/No prompting ('?' Prefix).....	18
YMD, READDATE format specifier	B-36
YYYYMMDD, date function	B-12
ZERODB, program file attribute variable.....	62
ZEROFILL, STRWRITE format specifier	B-38