LOG-ON PROCEDURE

ACOUSTIC COUPLER AND TELEPHONE

1. Set Terminal control switch to LINE.
2. Turn on coupler. If coupler has Duplex switch, set to FULL, and if coupler has line switch set to ON-LINE.
3. Dial System number. System will respond with high pitched tone. When the System answers, place hand-set in the coupler.
4. Press return key. System will respond with a colon.
5. Type in HELLO command. Example:

   :HELLO USER.ACCOUNT; TERM=6

   System will respond with session identification and another colon. If Error 9 (UNACCEPTABLE DEVICE) occurs, the terminal is not configured to accept sessions.
6. Type in the BASIC subsystem command.

   :BASIC

   (Assumes terminal as default for all command inputs, data inputs, and list outputs; to use other devices, see "MPE File Operations" in back.)
7. System will respond with subsystem identification and the BASIC prompt character >. Enter any valid BASIC command or statement.

DATA SET

1. Set Terminal control switch to LINE.
2. Push TALK button on Data Set.
3. Dial System number. System will respond with a high pitched tone. When the System answers, press the DATA button until the DATA light turns on; then replace the handset.
4. Same as 4, 5, 6, and 7 for Acoustic Coupler.

DIRECT CONNECTION TO THE SYSTEM

1. Set Terminal control switch to LINE.
2. Same as 4, 5, 6, 7 for Acoustic Coupler.

LOG-OFF PROCEDURE

1. Type EXIT to terminate BASIC subsystem.
2. Type BYE to terminate the session.
3. For telephone: replace hand-set.

TERMINAL TYPES

For use in :HELLO command. Default is 0.

0 HP 30124A (HP 2749B), ASR-33, or ASR-35
1 ASR-37
3 Execuport 300
4 HP 30123A (HP 2600A), or Datapoint 3300
5 Memorex 1240
6 HP 30120A (HP 2762A), GE Terminet 300, or DCT 500
7 IBM 2741 Selectric, CALL 360 Correspondence Code
8 IBM 2741 Selectric, PTTC/EBCD Code

For Selectric terminals refer to the MPE Operating System reference manual for special log-on procedure and other considerations.

SPECIAL CHARACTERS

NOTE: Superscript "C" indicates a control character (Hold control key down while striking appropriate character key).

FROM SYSTEM

: MPE prompt. An MPE command is expected.

> BASIC prompt. A BASIC command or statement is expected.

>> Continuation prompt. A continuation line is expected because preceding line ended with &. Do not enter a statement number.

? INPUT prompt. Input items are expected.

?? INPUT prompt. Further input items are required.

!!! Line delete. A full line was deleted by using X^C.

\ Character delete. A single character was deleted by using H^C.

??? Command error. A BASIC command was mistyped; retype correctly.

TO SYSTEM

H^c Character delete. May be repeated for multiple characters.

X^c Line delete. Delete the line being typed.

Y^c Subsystem break. Terminates execution of commands other than RUN. If program is running, it is suspended and control reverts to Interpreter; certain commands (see page 9 for list) may then be entered, including RESUME (or GO), which returns control to the program, and ABORT, which restores normal Interpreter mode.

BREAK (Key) System break. BASIC Interpreter is suspended and control reverts to the operating system. To re-enter BASIC, type RESUME.

& Continuation character. Use to continue any statement (except REM) on next line. Prompts with >>. Do not re-enter statement number.

OPERATORS

ARITHMETIC

\+ Addition (or unary sign)
\- Subtraction (or unary sign)
* Multiplication
/ Division
** Exponentiation (^ also acceptable)
MOD Modulo

RELATIONAL

= Equals
<> Does not equal (# also acceptable)
< Is less than
> Is greater than
<= Is less than or equal to
>= Is greater than or equal to

Non-numerical values compared on ASCII numerical value.

LOGICAL

AND Boolean "and"
OR Boolean "or"
NOT Boolean complement

MIN/MAX

MIN Select lesser value
MAX Select greater value

STRING

\+ Concatenate

HIERARCHY

1 **
2 NOT
3 * / MOD
4 +-
5 + (concatenate)
6 MIN MAX
7 Any relational
8 AND
9 OR

Left to right for operators at same level. Use parentheses to override above order. Nested parentheses evaluated inward out.

COMMANDS

APPEND

APPEND PROG2

Appends program from user's library to current program. First statement number of appended program must be greater than last number in current program. File name may be any valid MPE file name. Cannot append RUNONLY- or FAST-saved programs.

>BASIC

Interrupts an input request (INPUT, ENTER, or LINPUT) and enters a second level of BASIC. Use EXIT to return to initial level; prompts with ?? marks; timing then re-starts for ENTER.

CATALOG or CAT

CAT
CAT ALL
CAT PROG1, RECSIZE

Prints name, type, and maximum length (in records) of BASIC files and BASIC programs in user's library. If ALL is specified, also includes ASCII and binary files. If program or file name is specified, information for only that program or file is printed. If RECSIZE is specified, also

lists record width in words. Types are: BF = BASIC File, SP = Saved Program, FP = Fast-saved Program, A = ASCII, B = Binary. Can stop listing with Y^C. To catalog files not in user's log-on group, or to list on device other than terminal, or to begin listing at specified file, consult BASIC/3000 Reference Manual. Full syntax is:

```
CAT [fileset] [,ALL] [,RECSIZE] [,OUT=asciifile]
                                [,START=filename]
```

DELETE or DEL

```
DEL 100, 510, 400-420
```

Deletes individual statement or inclusive range of statements.

EXIT

Terminates BASIC subsystem and returns control to operating system. Also used to return from second BASIC level (see >BASIC).

GET

```
GET PROG1
```

Loads specified program from user's library, replacing current program in user's work area.

KEY

Returns control to keyboard after TAPE or XEQ mode.

LENGTH or LEN

Prints length (in 16-bit computer words) of program currently in user's work area.

LIST

```
LIST
LIST 50-100
LIST 140
LIST 1-2000,OUT=PRINTER,RECSIZE=132,
                                NONAME
```

Lists all of current program, or only those statements in a specified range, or only an individual statement. Listing may be directed to any ASCII file with OUT=; specify RECSIZE if other than default of 72 characters per record. NONAME deletes word "NAME" from start of listing.

NAME

```
NAME PROG3
```

Assigns a name to current program. One to eight alphabetic/numeric characters, beginning with alphabetic.

PUNCH

```
PUNCH
PUNCH 1-100
PUNCH 140
```

Punches all of current program, or only those statements in a specified range, or only an individual statement, on paper tape. Similar to LIST, except PUNCH listing is preceded and followed by ASCII null characters for paper tape leader and trailer. Full syntax is:

```
PUNCH [first[-last]] [,OUT=asciifile]
    [,RECSIZE=integer] [,NONAME]
```

PURGE

```
PURGE PROG1
```

Deletes specified program or file from user's library. Current program is not affected.

RENUMBER or RENUM

```
RENUM
RENUM 1000
RENUM 1000, 5,
RENUM 1000, 5, 100
RENUM 1000, 5, 100-999
```

Without parameters, renumber all statements in current program beginning at 10, with intervals of 10. First parameter, if given, specifies new starting point. Second specifies new interval. Third specifies beginning point of old sequence. Fourth specifies ending point of old sequence.

RUN

```
RUN
RUN,60
RUN PROG1, 100
```

Without parameters, starts execution of current program. With comma, integer: starts execution at specified statement. With program name: retrieves and starts execution of named program, optionally starting at specified statement. For other parameters, consult BASIC/3000 Reference Manual. Full syntax is:

```
RUN [progname] [,label] [,OUT=asciifile]
    [,NOWARN] [,FREQ] [,NOECHO]
```

(FREQ prints a "frequency of execution" table of each statement in the program, after program terminates.)

SAVE

SAVE
SAVE PROGX
SAVE PROGX!,FAST,RUNONLY

Creates a new file in user's library and stores current program in it. Program must be named; if named by SAVE, name is given to stored copy only. If another file has same name, command is rejected unless name is followed by exclamation mark, which purges old file. FAST option causes program to be checked for validity before SAVE; FAST-saved programs are faster to GET, CHAIN, and INVOKE. RUNONLY limits allowable commands (see table, page 9).

SCRATCH or SCR

Deletes current program and its name from user's work area.

SPOOL

Enables programs (with data) and/or commands to be read from paper tapes not having X-OFF character. Type SPOOL; turn on reader. Tape is read and commands, if any, are executed. Type Y^{C} to end tape mode.

SYSTEM

Suspends BASIC and passes control to operating system. Use :RESUME to return to BASIC.

TAPE

Enables programs (with data) and/or commands to be read from a paper tape reader. Tape must have X-OFF character terminating each record. Type TAPE; turn on reader. Tape is read and commands, if any, are executed. Type KEY (or include as last command on tape) to end tape mode.

XEQ

XEQ BATCHJOB, ECHO

Specifies a new command file. Upon end of file, Y^{C}, or a KEY command, control reverts to original command file. Optional ECHO parameter echoes commands to list file. (XEQ file may be generated with EDIT/3000, KEEP unnumbered.)

Legal Times for Command Input

	NORMAL	DURING BREAK	SAVED RUNONLY		NORMAL	DURING BREAK	SAVED RUNONLY
CATALOG	X	X	X	UNTRACE	X	X	
CREATE	X	X	X	GET	X		X
DUMP	X	X	X	RUN	X		X
EXIT	X	X	X	SCRATCH	X		X
KEY	X	X	X	APPEND	X		
PURGE	X	X	X	DELETE	X		
SPOOL	X	X	X	NAME	X		
SYSTEM	X	X	X	RENUMBER	X		
TAPE	X	X	X	SAVE	X		
XEQ	X	X	X	Statements	X		
BREAK	X	X		ABORT		X	X
LENGTH	X	X		RESUME, GO		X	X
LIST	X	X		CALLS		X	
PUNCH	X	X		FILES		X	
TRACE	X	X		SET		X	
UNBREAK	X	X		SHOW		X	

NORMAL: Prior to RUN

DURING BREAK: Break caused by debug BREAK or Y^{C}

SAVED RUNONLY: Prior to RUN of (or during Y^{C} break in) a program SAVEd with RUNONLY option

STATEMENTS

CALL or *

```
10 CALL LOR (X,Y,Z)
20 *WHOM (U$,G$,A$(*,*))
```

Transfers control to specified procedure or subprogram in segmented library (SL). CALL and asterisk are equivalent. Parameters are passed by reference, not by value. Arrays may be passed and returned; numeric array name must be followed by one or two asterisks in parentheses to denote one or two dimensions; two asterisks for string array.

CHAIN

```
30 CHAIN "PROG2", 200
```

Loads and runs named program, beginning at optional statement number specified. Current program is deleted except for COMmon variables; current program's files are closed.

```
40 CHAIN A$(7,12), 8*X
```

Program name can be given by any string expression. Statement number can be any integer expression.

COM

```
2 COM A,B1, C(20), D$(3,2)
```

Creates storage space (COM block) in user's work area for common variables; they can then be accessed by other COM statements in invoked or chained programs. Must precede all DIM, DEF, or type statements in program. Arrays and strings must be dimensioned.

```
3 COM(7) B2, INTEGER J,K, LONG L1
```

Ten COM blocks allowed, one unlabeled plus COM(1) thru COM(9). Numeric values assumed to be real unless preceded by type specifier; type specifier applies until end of list, next specifier, or next string variable.

```
2 COM  A, X1, V(20), U$(*,*)
```

COM statements in subsequent programs must agree in order, type, and dimension. Assigned names may differ. Asterisk means assume original dimension (valid only if COM block was made active by invoking program).

```
4 COM(3) M(6,2), N(4,5), R7, Q$(8)
5 COM(3) O(12,3), P(*)
```

More than one COM statement may reference same block provided the statements are consecutive; storage space is concatenated (example shows 6 elements in COM(3)).

COMPLEX

```
50 COMPLEX A, B(3,3)
```

Allocates space for type complex variables and arrays.

DATA

```
60 DATA 3, 8.4, "ABC", (3.6,-12.9)
```

Provides data which is read left to right. Data pointer initially points to first item in first DATA statement; advanced or repositioned by READ and RESTORE. DATA statements may be located anywhere in program, though considered to be continuous list of data. Strings must be enclosed in quotes; complex numbers in parentheses.

END

```
70 END
```

Terminates program execution. Not specifically required since implicit END follows last line of any program. In an invoked program, causes a return to invoking program at statement following INVOKE; unlike STOP.

ENTER

```
 80 ENTER #T
 90 ENTER #T,A,R,X
100 ENTER A,R,S$
```

Fills variable T with logical device number of user's terminal and/or allows the user a specified number of seconds to reply (A=1 to 255), returns actual response time (R), and returns the value entered (X, S$). Response time negated for wrong data type input; -256 for time out. Any expression may be substituted for variable A. No prompt given. No line feed following carriage return.

FOR-NEXT

```
110 FOR A=1 TO 5
120    FOR B=-1 TO -3 STEP -1
130       PRINT B
140    NEXT B
150 NEXT A
```

Repeats statements between matching FOR and NEXT statements, modifying the loop variable (A,B) over a range of values, starting with a specified initial value to a final value (inclusive), stepping by a specified or default (1) step value. FOR and STEP values may be any expression, except STEP should not be zero.

GOSUB

```
160 GOSUB 1000
165 STOP
1000 REM: START OF SUBROUTINE
1200 RETURN
```

Transfers control to beginning of simple subroutine at specified statement number. RETURN in subroutine returns control to statement following GOSUB. May be nested to 10 levels.

```
170 GOSUB N OF 1000, 1210, 1320
```

Multibranch GOSUB evaluates integer expression N and branches to Nth statement number in the list. GOSUB is ignored if expression is less than 1 or greater than the number of items in the list.

GOTO

```
180 GOTO 200
190 GOTO N OF 210, 230, 250
```

Transfers control to specified statement number. Multibranch GOTO evaluates integer expression N and branches to Nth statement number in the list. GOTO is ignored if expression is less than 1 or greater than the number of items in the list.

IF-THEN

```
200 IF A<0 THEN 400
210 IF A=B THEN PRINT B
220 ELSE A=-A
```

Evaluates expression following IF; if true, transfers control to specified statement number following THEN, or executes clause statement following THEN. If false, clause statement is ignored; may be followed by ELSE, which specifies a clause statement to be executed or a statement number to branch to; ELSE is skipped if IF was true. Expression resulting in nonzero is considered true; zero considered false. Clause statement may not be IF or a declaration.

```
230 IF A<100 THEN DO
235    X=A
240    PRINT A
250 DOEND
260 ELSE DO
265    X=B
270    PRINT B
280 DOEND
```

DO/DOEND pairs enclose conditionally executed groups of statements. In example: 235 and 240 are executed if expression is true, 265 and 270 if false. Additional IF-THENs may be nested.

INPUT

```
300 INPUT A,B$,C(2,2)
```

Prints out question mark on new line, then accepts numeric or string inputs from terminal (or inputfile assigned in :BASIC command). Each input constant, separated by commas, is assigned in turn to each named variable and must agree as numeric or string. (Numeric type conversion is automatic.) If insufficient constants are input, double question mark (??) prompts user for more data. String inputs must be in quotes unless only or last item in data input.

```
310 INPUT (FOR N=1 TO 15, A(N))
320 INPUT (FOR N=1 TO 15, (FOR M=1 TO 5,
                                B(N,M)))
```

FOR loop iterates sections of input list. Example: to fill one- or two-dimensioned array as each item is input.

```
330 INPUT "ANSWER IS: ", C$
340 INPUT (FOR N=1 TO 15, "MORE", A(N))
```

Quoted string is printed as a prompt, instead of question mark, for the immediately following variable. Prompt string is printed only if following variable is first to be input on a new line.

```
350 INPUT : A,B,C:
360 INPUT:
```

Trailing colon after item list saves excess inputs in buffer; omission of trailing colon clears buffer. Leading colon uses data from buffer for filling named variables before prompting; omission of leading colon clears buffer before prompting. INPUT with colon alone puts entire input line into buffer.

INTEGER

```
370 INTEGER A, B(3,3)
```

Allocates space for type integer variables and arrays.

INVOKE

```
380 INVOKE "PROG2",200
```

Retrieves and runs named program, beginning at optional statement number specified. Current program is suspended, and its execution resumes at end of invoked program. Files remain open and COMmon variables can be accessed.

```
390 INVOKE A$(7,12), 8*X
```

Program name can be any string expression. Statement number can be any integer expression.

LET

```
400 LET A=3.5, A$= "ABC", N=N+1
410 X=Y+X/10
420 R=S=T=5
```

Assigns value (or value of expression) on right side of assignment operator (=) to variable on left side. Word LET is optional. Multiple assignment permitted.

LONG

```
430 LONG A,B(3,3)
```

Allocates space for type long variables and arrays.

PRINT

```
400 PRINT A,5,8/E; B[4]; C$ "Q=9*",
```

Prints value of specified expression. PRINT by itself skips one line. If there is insufficient space to print all items on one line, remaining items are printed on next succeeding line. Comma spaces items 5 fields per line (15, 15, 15, 15, 12 characters). Semicolon spaces numeric items on multiple-of-3 column boundaries; semicolon preceding a string expression suppresses spacing. Separator may be omitted between quoted string and another item; in this case it is equivalent to a semicolon. Comma or semicolon at end of line suppresses final carriage return and line feed.

```
405 PRINT (FOR N=1 TO 15, A(N))
410 PRINT (FOR N=1 TO 15, (FOR M=1 TO 5,
                                B(N,M);))
```

FOR loop iterates sections of print list. Example: to print out contents of one- or two-dimensioned array.

```
420 PRINT "NAME" SPA(3) "ADDRESS";
425 PRINT CTL(49); LIN(3); Y$; TAB(R+4), D$
```

Print function SPA inserts specified number of spaces in printed line. CTL writes preceding data with carriage control according to Carriage Control Codes table (primarily intended for line printers, etc.). LIN generates carriage return and specified number of line feeds; negative value suppresses carriage return; zero value causes carriage return only. TAB moves carriage to specified print position; range 0 to 71. Any valid expression which evaluates to an integer may be used in SPA, LIN, CTL, and TAB functions. A comma after any print function is treated as a semicolon.

Carriage Control Codes

Decimal Code	Carriage Action
32	Single-space
43	Carriage return, no line feed
48	Double-space
49	Page eject (form feed)
64	Post-spacing
65	Pre-spacing
66	Single-space, with auto page eject (60 lines/pg)
67	Single-space, without auto page eject (66 lines/pg)
128+nn	Space nn lines (no automatic page eject). nn=1 thru 63 (i.e., codes 129 thru 191).
192	Page eject (*ftc #1)
193	Skip to bottom of form (*ftc #2)
194	Single-spacing, with auto page eject (*ftc #3)
195	Single-space on next odd-numbered line, with auto page eject (*ftc #4)
196	Triple-space, with auto page eject (*ftc #5)
197	Space 1/2 page, with auto page eject (*ftc #6)
198	Space 1/4 page, with auto page eject (*ftc #7)
199	Space 1/6 page, with auto page eject (*ftc #8)
208	No carriage return, no line feed.
256	Post-spacing
257	Pre-spacing
258	Single-space, with auto page eject (60 lines/pg)
259	Single-space, without auto page eject (66 lines/pg)
	*Format Tape Channel number

Format String Elements

Variables		EXAMPLES
A	ASCII character	AAA
D	Decimal digit	DDD
Blanks		
X	Blank characters	XXX
Separators		
,	Separator only	AA,DD
/	Carriage return, line feed	AA///D/
Replicators		
n	Single replicator	3A
n( )	Group replicator	3(3A,2X,3D)
Numeric Specifications		
M	Minus sign	MDD
S	Sign character	SDD
.	Decimal point	DD.D
E	E-type floating format	DD.DE
C	Complex format	C(3D,M2D)
+ or –	Sign position for imaginary part	DD+DD
Literals		
" "	Literal string	"APRIL"
$	Dollar character	$D.DD
I	Alphabetic "I"	DDXI
Compressed Format		
K	Compress specifier	"ONLY",XKX,"ITEMS"

Carriage Control		
+	Suppresses line feed	+,6(4DX)
–	Suppress carriage return	-,16A
#	Suppresses carriage return and line feed	#,6(6A)

PRINT USING

```
430 J$="DD.DD,X3D"
435 PRINT USING J$; A,852
440 PRINT USING "3A,3A"; A$, "ABC"
445 PRINT USING 450; A$, "ABC", A,852
450 IMAGE "JANUARY" ///2(3A), $DD.DD,X3D
```

Prints data items using format specified before semicolon. Commas are the only legal separator in data list. Format is specified one of three ways: string variable referencing format string (435), or format string in quotes (440), or statement number referencing an IMAGE statement which contains format string (445). If end of format string is reached before end of data list, processing returns to beginning of string.

Format Strings

See table, "Format String Elements". In format string, A specifies ASCII character, D specifies digit, X specifies space. Comma separates specifications. Slash separates specifications and causes carriage return and line feed. Single replicator may precede A,D,X,I,$. Group relicators may be nested indefinitely.

M indicates placement of "-" only; S indicates placement of "+" and "-" signs. Decimal point position is indicated by period. E following an integer or fixed specification computes and prints an exponent field. For complex numbers, imaginary part is dropped if floating point specification is used. To also print imaginary part, precede imaginary specification with + (equivalent to S) or - (equivalent to S if number is negative, blank if not). Or use C format, enclosing both real and imaginary specifications (separated by comma) within parentheses and preceding with a C.

Literal strings are valid in format string only if given in IMAGE statement. Other literals (I and $) are valid anywhere.

K specifier inserts numeric value using only enough character positions to print the value (deletes blanks, trailing zeros and decimal point). May be preceded or followed by X, $, or I, with or without replicators.

Carriage control character; if used, must be first in format string. Must be followed by comma. Takes effect after printing.

READ

```
500 READ X,A$ (FOR J=1 TO 10,B(J))
510 READ (FOR J=1 TO 15, (FOR K=1 TO 5,C(J,K)))
```

Reads numeric or string constants from DATA statements and assigns constants to named variables. FOR loop in parentheses iterates sections of read list. Example: to read data into one- or two-dimensioned array. READ begins reading at current setting of data pointer (see DATA and RESTORE) and advances pointer after each read. Strings must be read into string variables, numbers into numeric variables.

REAL

```
520 REAL A,B(3,3)
```

Allocates space for type real variables and arrays. Generally used to define local variables in multiline user functions. Not usually required elsewhere because REAL is default type.

REM

```
530 REM --ANY TEXT--
```

Allows placing remarks within program listing. Cannot be continued with & symbol. All characters may be used, including: "&/*.

RESTORE

```
540 RESTORE 60
```

Resets data pointer to DATA statement at the specified statement number (or nearest following one if specified statement is not DATA). If statement number omitted, resets to lowest numbered DATA statement.

RETURN

```
550 RETURN
```

Returns control from a GOSUB subroutine to statement immediately following last GOSUB executed.

STOP

```
560 STOP
```

Terminates present run, including any invoking programs, and returns control to BASIC.

SYSTEM

```
570 SYSTEM T, "FILE ABC; DEV=TAPE"
```

Allows execution of any valid MPE operating system command. Command given as a string expression. System returns 0 to test variable if execution successful, MPE error code if not.

RULES FOR STRINGS

```
600 DIM A$(5), A1$(3)
610 LET A$="ABCDE", A1$="FG", B$="Q"
```

Strings are enclosed by quotes, may be up to 255 characters in length. String variable is one letter or one letter and one digit, followed by $. Simple strings and string arrays may not have same name (unlike numeric case). Strings longer than one character must be dimensioned by DIM or COM. Dimensioned length is maximum size or "physical length"; actual length of contents determines "logical length" (logical length of A1$ is 2). If LET assignment exceeds physical length, string is truncated on right.

```
620 PRINT '34 "LINE" '34 '10 '13
```

An individual character may also be specified by apostrophe preceding decimal number which represents an ASCII character. (See ASCII table; '34 is quote mark, '10 is line feed, '13 is carriage return.)

```
630 C$ = A$(3)
640 D$ = A$(3,5)
650 E$ = A$(3;3)
```

Substrings are specified by one or two numbers (or expressions) in parentheses. One number specifies a beginning character; substring extends to logical length of string. Two numbers separated by commas specify beginning and ending character positions respectively. Two numbers separated by semicolon specify beginning character position and quantity of characters respectively. If A$ is a source string of 5 characters, C$, D$, and E$ are identical.

```
660 DIM R$(5,12)
670 READ (FOR N=1 TO 5, R$(N))
680 PRINT R$(1)
690 F$ = R$(2,6)
700 G$ = R$(2,6,12)
705 H$ = R$(2,6;6)
```

String arrays are one-dimensional only. (Statement 660 dimensions a 5-element array of 12 characters each; 670 fills the array.) Subscript specifies element number (680). Substrings are specified by one or two additional subscript designators, using the same conventions as described above for simple strings.

Strings can be compared by relational operators on ASCII numerical basis, and can be concatenated by + operator. (See OPERATORS.)

CONVERT

710 CONVERT X TO A$(3)

Converts the numeric value of X into printable ASCII (without trailing blanks, zeros, and decimal point) and deposits it in specified string destination (third element of array A$).

720 CONVERT C$ TO Y(1), 200

Converts string into its numerical format and deposits it in specified numeric variable (first element of array Y); transfers to optionally specified statement (200) in case of error.

DIM

730 DIM C$(30), A$(5,10)

Reserves storage and declares name for simple strings and string arrays. For string arrays, both array length and string length are specified. Maximum length of C$ is 30 characters. Maximums for A$: 5 elements of 10 characters each.

LINPUT

740 LINPUT B$(2)

Entire line of input is accepted as a single string. May be assigned to simple variable, array element (as in example), or substring. All characters accepted. No prompt given.

REDIM

750 REDIM A$(3), C$(10)

Redimensions specified string arrays. Cannot increase beyond original dimensions. Does not alter maximum string length specification. Dimension may be integer expression.

FUNCTIONS

String related functions listed below accept string values as parameters and/or return a string value as their result. The result may be embedded in other numeric, string or logical expressions.

Function	Returns
CHR$(65)	Character equal to ASCII decimal equivalent
DAT$(1,27)	Current date and/or time
DEB$(A$)	String, leading and trailing blanks deleted
LEN(A$)	Logical length of string
NUM(A$)	ASCII decimal equivalent of first character
POS(A$,B$)	Starting position of substring in A$ which = B$
ROW(A$)	Number of elements in array
UPS$(A$)	String, all lower case letters upshifted
WRD(A$,B$)	Like POS, but substring cannot be preceded or followed by a letter

Format of DAT$ String

Char
- 1-3: Day of the week (SUN, MON, TUE, WED, THU, FRI, SAT)
- 4-5: Comma and blank
- 6-8: Month of the year (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC)
- 9: Blank
- 10-11: Day of the month (1 to 31)
- 12-13: Comma and blank
- 14-17: Year
- 18-19: Comma and blank
- 20-21: Hour (1 to 12)
- 22: Colon
- 23-24: Minute (0 to 59)
- 25: Blank
- 26-27: AM or PM

ARRAYS

RULES FOR ARRAYS

```
10 COM A1(100), A(2,15), A$(12,5)
810 DIM B1(100), B(2,15), B$(12,5)
820 INTEGER E(50,50)
830 READ D1(5), D(2,5)
```

Storage for arrays is allocated by COM or DIM statement, type statement, or default usage. Default array is 10 elements for one-dimensional, 10 x 10 elements for two-dimensional; numeric only. Array variable is one letter or one letter and one digit; followed by $ for string array. Name of numeric array variable may be same as a simple variable elsewhere in program. In dimensioning statement (first three above) the first or a single number in paren-

theses declares number of rows (horizontal rows when printed); second number declares columns for numeric arrays or maximum string length for string arrays. String arrays are one-dimensional only. Dimensioned maximum size of array is "physical size"; "logical size" is the number of rows, or rows and columns, actually filled.

```
830 READ D1(5), D(2,5)
840 LET A1(25)=36.5, A1(26)=321E-01
850 INPUT (FOR N=1 TO 2, (FOR M=1 TO 15,
                              A(N,M)))
```

Arrays may be filled by READ, LET, and INPUT statements in addition to MAT statements defined below. First number in parentheses (or first FOR loop in line 850) defines row; second number (or second FOR loop) defines column. Mixed numeric data types may be entered into array but are converted to array's data type; default is type real.

DIM

```
810 DIM B1(100), B(2,15), B$(12,5)
```

Declares maximum size for one- or two-dimensioned numerical arrays (type real), or maximum array length and string length for string arrays. All arrays must be dimensioned in DIM, COM, or type statement, or assume default size of 10 or 10 x 10 elements.

MAT

```
860 MAT C = A
865 MAT B$ = A$
870 MAT C = A+B
880 MAT C = A-B
890 MAT C = A*B
900 MAT C = (2.5)*B
```

Stores result of specified matrix operation into destination array, left of assignment operator (=). Same array may appear on both sides. Destination array must be dimensioned large enough to accept result; automatically redimensioned to fit result. Example 860 copies array A into array C. Example 865 copies string array A$ into B$ (truncating string elements if necessary). Example 870 adds each corresponding element of two arrays, having identical logical size, and stores the result in C. Example 880 subtracts elements in B from elements in A, and stores the result in C; A and B must have identical logical size. Example 890 multiplies elements as follows: each element of first row of A multiplied with each element of first column of B; sum of products becomes C(1,1); same row of A multiplied with each element of second column of B; sum products becomes C(1,2); continue until all rows of A are multiplied by all columns of B; column dimension of A must equal row dimension of B; for dimensions of C, use row dimension of A and column dimension of B. Example 900 multiplies all elements of B by the numeric expression in parentheses.

MAT CON

```
910 MAT A = CON
915 MAT B = CON(3,4)
```

Sets all elements of array to 1. Optionally redimension.

MAT IDN

```
920 MAT C = IDN
925 MAT D = IDN(3,3)
```

Sets identity pattern into matrix; i.e., elements on major diagonal (upper left to lower right) set to 1, all others to 0. Optionally redimension. Array need not be square.

MAT INPUT

```
930 MAT INPUT E
935 MAT INPUT E(3,4)
```

Prompts with ? (or ?? second time and thereafter), then accepts matrix elements from terminal. Optionally redimension. Also valid for string arrays, but can redimension only the array length, not string length. Colon extension for input buffering and embedded FOR loops not allowed.

MAT INV

```
940 MAT G = INV(H)
```

Assigns inverse of a square array to destination array, with automatic redimensioning. (Inverse of an array is one which, when multiplied by original array, results in identity array.) Not valid for integer arrays.

MAT NUL$

```
945 MAT A$ = NUL$(10)
```

Sets each element of specified string array to null string. If optional integer expression in parentheses is specified, also redimensions number of elements in array.

MAT PRINT

```
950 MAT PRINT A,LIN(2),B; C$
```

Prints contents of specified arrays. Comma after array name spaces elements as if separated by commas; semicolon packs elements as if separated by semicolons. No

punctuation after last array assumes comma, but any trailing punctuation also suppresses final carriage return and line feed. Prints with double spacing between rows. One-dimensional array printed as single row.

MAT PRINT USING

960 MAT PRINT USING "5(DXDDXX)/"; A
965 MAT PRINT USING 450; B
970 MAT PRINT USING J$; C$

Prints contents of arrays specified after semicolon using format specified before semicolon. Format specifications given under "PRINT USING".

MAT READ

975 MAT READ A, B(3,4), C$

Reads numeric or string constants from DATA statements and stores into successive elements of named arrays. Begins reading at current setting of data pointer, and reads until array is full. Can redimension. For string arrays, can redimension only the array length, not string length.

MAT TRN

980 MAT A = TRN(B)

Copies contents of array B into array A with rows and columns transposed. May be the same array.

MAT ZER

985 MAT A = ZER(3,4)

Sets all elements of array to 0. Optionally redimension.

REDIM

810 DIM B1(100), B(2,15), B$(12,5)
990 REDIM B1(75), B(3,10), B$(10)

Redimensions specified arrays. Cannot change number of dimensions or increase total number of elements beyond original dimensions. For string arrays, can redimension array length only, not string length. Integer expression may be used for dimension.

FUNCTIONS

995 PRINT ROW(B1), COL(B)

ROW returns number of rows in specified array. COL returns number of columns.

ADVANCE

100 ADVANCE #3;5,X

Advances file pointer of specified file (#3); integer or integer expression specifies number of data items to be skipped (5). Negative integer decrements pointer. Result value is returned to numeric variable (X): 0 if successful, difference between requested and actual advance if end-of-file or start-of-file encountered; negative value for reverse direction. For BASIC formatted files only.

ASSIGN

110 ASSIGN "DFILE", 6, D1
120 ASSIGN X$, 8, X, "ABZ1", RR
130 ASSIGN *, 16

Assigns a file name to a file number previously reserved by FILES, and opens the file; or (*) closes specified file number. File name can be specified by literal string or string variable ("DFILE", X$). File number is an integer or integer expression of value 1 to 16. Numeric variable (D1, X) receives result value of ASSIGN (see "ASSIGN Parameters" table). Either or both a data scrambling mask (string in quotes) and an access restriction code (see "ASSIGN Parameters" table) may be included.

ASSIGN Parameters

Result Value

0	File available for read and write
1	File available for read only
3	File does not exist
5	No buffer space for file
6	File access restricted (RR) by another user
7	Requested restrictions not possible
8	File available for write only

Access Restriction

RR	Read and write restriction
WR	Write restriction
WL	Write restriction, LOCK allowed
NR	No restriction
NL	No restriction, LOCK allowed

CREATE Command
CREATE Statement

```
CREATE AFILE,30,212
140 CREATE B1, "BFILE", 15, 319
150 CREATE B2, C$, 15
```

Creates a BASIC formatted file, assigns its name, and declares its structure. Statement form also returns result value. File name is unquoted string for command form, and quoted string or string variable for statement form. Name may be fully qualified:

```
"AFILE/LOCKWORD.PUB.ACCTNAME"
```

First integer value after name specifies number of records in file; can be integer expression for statement form. Second integer value, if present, specifies words per record from 4 to 319 (suggest 106, 212, or 319); can be integer expression for statement form; default is 106. Numeric variable (B1, B2) returns:

0 successful file creation
1 file exists with same name
2 unsuccessful, other reason

Formula to determine physical record size created, knowing user-assigned logical record size:

```
P = R+INT(R/5)+1
```

To determine logical record size, knowing physical record size created:

```
R = CEI(5*(P-1)/6)
```

DUMP Command

```
DUMP AFILE
DUMP BFILE, OUT=L
```

Prints contents of a BASIC formatted file on terminal or, using OUT= parameter, on a specified ASCII file (e.g., line printer). Each item is printed on a separate line. Record boundaries not indicated. Stops on first end-of-file.

FILES

```
160 FILES AFILE,*,#4
```

Assigns or reserves a local file number (1 to 16) for files to be accessed in this program. Numbers determined by ordinal position in statement. Statement may be located anywhere in program. If more than one FILES statement given, file numbers begin with first FILES statement. If file designator is a file name (may be fully qualified — see CREATE), file is also opened; only read access granted if

already opened by another program. If file designator is *, file number (2 in example) is reserved for later assignment by ASSIGN. If file designator is #integer, file specified by internal file number (#4) is opened and assigned a local file number (3). Internal file numbers begin with 1 in main program and end at last assigned local file number; then either: continue on with any numbers which may be assigned by FILES in an INVOKEd program, or, for FILES in a CHAINed program, begin again with same numbers assigned in the calling program. Valid for binary, ASCII, and BASIC formatted files.

LINPUT

```
170 LINPUT #8;A$
175 LINPUT #10,2;B$
```

Reads contents of entire record into string variable. Taken from specified file at current position of file pointer (170), or from specified file and record (175). Extra characters discarded if string dimension not large enough. For ASCII files only.

LOCK

```
180 LOCK #8
```

Provides cooperative file access control for multiple users. If LOCK is successful, no other program can then lock; if already locked, program suspends until other program UNLOCKs. Valid only for files opened by ASSIGN with WL or NL restriction.

MAT PRINT

```
190 MAT PRINT #4;A,B$
200 MAT PRINT #4,2;A,B$,END
```

Writes contents of entire arrays into specified file, either serially (beginning at current position of file pointer) or direct (into specified record). Array may cross record boundaries in either case. If followed by END, writes end-of-file after last array.

MAT READ

```
210 MAT READ #4;A,B$
220 MAT READ #4,2;C,D$
```

Fills contents of specified arrays from a file, beginning either at current position of the file pointer or at a specified record. Reads until arrays are full; may cross record boundaries.

ON END

```
230 ON END #4 THEN 500
```

Causes transfer to specified statement if end-of-file condition occurs in subsequent reading or writing of specified file. Transfer label can be altered by subsequent ON END. IF END is equivalent to, and stored internally as, ON END.

PRINT

```
240 PRINT #5; A, B$
250 PRINT #5,2; A, B$. END
260 PRINT #6,1
```

Writes specified numeric or string values into specified file, either serially (beginning at current position of file pointer) or direct (starting at beginning of specified record). Data may cross record boundaries in either case, but single item cannot be larger than record size (except for binary files). If followed by END, writes end-of-file after last data item; end-of-file may be overlaid by subsequent PRINT. Direct PRINT without print values erases contents of specified record.

PURGE Command
PURGE Statement

```
PURGE A FILE
270 PURGE X, "BFILE"
```

Deletes specified binary, ASCII, program, or BASIC formatted file from system. Statement form returns result value to numeric variable (X):

0 successful purge
1 file being accessed
2 not permitted to purge
3 no such file

READ

```
280 READ #5;A,B$
290 READ #5,2;C,D$
300 READ #5,3
```

Reads items into numeric or string variables from a file, beginning either at the current position of the file pointer or at a specified record. Reads until item list is exhausted; may cross record boundaries. Direct READ without variables list positions file pointer to beginning of specified record.

RESTORE

```
310 RESTORE #2
```

Repositions file pointer to beginning of file. Rewinds magnetic tape. (Direct access not required.)

UNLOCK

```
320 UNLOCK #8
```

Unlocks LOCKed file. (Ensures that last PRINT# operation is physically complete first.)

UPDATE

```
330 UPDATE #3; 4.57
```

Replaces data item last read in specified file with data item given. For numeric replacement, converts value of expression to type of former item. For string replacement, new string is truncated or blank-filled on right to fit size of former item. For BASIC formatted files only.

FUNCTIONS

```
340 PRINT TYP(1), TYP(-2)
350 PRINT REC(3)
```

TYP function returns a value that defines the data type of the next item in the specified file — i.e., at the position of the file pointer. Negated file number specifies direct file; end-of-record marks are reported. Values are:

1 real
2 string
3 end-of-file
4 end-of-record
5 integer
6 long
7 complex

REC function returns record number at which the file pointer for a specified file is currently positioned. File number for TYP or REC may be given as an integer expression.

FUNCTIONS

MATH FUNCTIONS

These functions return a numeric result; may be used as expressions or parts of expressions. PRINT used to illustrate general form only; other statement types may be used. Arguments for trigonometric functions must be expressed in radians. Type real or type integer arguments return real result; for type long or type complex, consult BASIC/3000 Reference Manual.

```
400 PRINT ABS(X),ATN(X),CEI(X)
```

ABS(X)	gives absolute value of X.
ATN(X)	gives arctangent of X.
CEI(X)	gives ceiling of X.
CNJ(X)	reverses sign of imaginary part of X.
COS(X)	gives cosine of X.
CPX(X,Y)	makes complex number of X and Y.
CSH(X)	gives hyperbolic cosine of X.
EXP(X)	gives e raised to power of X.
IMG(X)	gives imaginary part of X.
INT(X)	gives largest integer $\leq$ X.
LOG(X)	gives natural logarithm of X.
PIX(X)	gives product of π times X.
REA(X)	gives real part of X.
SGN(X)	gives -1, 0, or 1 for sign of X.
SIN(X)	gives sine of X.
SNH(X)	gives hyperbolic sine of X.
SQR(X)	gives square root of X.
TAN(X)	gives tangent of X.
TNH(X)	gives hyperbolic tangent of X.

OTHER FUNCTIONS

```
410 Y=BRK(0)
500 Y=BRK(1)
```

BRK with argument =0 disables >BASIC, the Break key, and Y^C break. If argument is >0, these functions are enabled. Returns 0 if breaks were previously disabled, 1 if enabled. (Argument <0 returns current setting only.)

```
420 PRINT BUF(0)
```

BUF determines data type of next item in input buffer. Argument is ignored. Values returned:

1 real	5 integer
2 string	6 long
4 empty buffer	7 complex

```
430 PRINT CPU(0)
```

CPU returns number of seconds of CPU time (±.001 sec.) that the program has run. Argument is ignored.

```
440 PRINT RND(0)
450 PRINT RND(-1), RND(0), RND(0), RND(0)
460 GOTO 450
```

RND with argument ≥0 returns random number between 0 and 1 but not =1. To generate repeatable sequence, make first call with argument <0, subsequent calls >=0. To repeat sequence, use argument value from first call.

```
470 PRINT TIM(-1), TIM(1); TIM(0)
```

TIM returns a time value determined by argument:

<0	number of wall-time seconds since program began
0	current minute (0-59)
1	current hour (0-23)
2	current day of year (0-366)
3	current year (0-99)

```
480 PRINT TYP(0)
```

TYP with argument =0 returns a value that defines the data type of the next item in a DATA statement – i.e., at the current position of the data pointer. Values are:

1 real	5 integer
2 string	6 long
3 end of data	7 complex

```
490 PRINT UND(A)
```

UND returns 1 if specified numeric variable is undefined, 0 otherwise.

ONE-LINE USER FUNCTIONS

```
510 PRINT FNA (5,2.5)
520 DEF FNA (X,Y)=(X*Y)+(X/Y)
```

Numeric function returns numeric value to the calling statement. Operation: actual parameters in calling statement (5, 2.5) are passed to corresponding formal parameters (X, Y) in DEFinition statement; function is executed using passed parameters and numeric result replaces entire function call; equivalent to PRINT 14.5. Function names are FNA thru FNZ. Formal parameter names may duplicate variable names used elsewhere in program.

```
530 PRINT FNB(M,N(2))
540 DEF INTEGER FNB(R,INTEGER S)=R*S+J
```

Actual parameters (M, N(2)) may also be given as simple or subscripted variables. Variable names need not be same

as formal parameter names (R, S); however, quantity of parameters must be identical. Function defined by DEF may include, or consist entirely of, constants and variables not passed by function call (J); if no parameters passed, use dummy variable in parentheses. To return result in other than type real, precede function name in DEF with desired type; data types converted automatically.

```
550 X$="CD",Y$="GH"
560 PRINT FNA$("AB",X$)
570 DEF FNA$(R$,S$)=R$+S$+"EF"+Y$
```

String function returns string value to calling statement; equivalent to PRINT "ABCDEFGH". Function names are FNA$ thru FNZ$. Actual parameters in calling statement may be quoted string, string expression, or string variable. String function in DEF may include passed parameters, quoted strings or string expressions, and string variables not passed by function call (Y$).

MULTILINE USER FUNCTIONS

```
600 PRINT #1; "ONLY"+FNA$(5)+"ITEMS"
610 DEF FNA$(N)
620     DIM S$(12)
625     CONVERT N TO S$
630     RETURN S$
640 FNEND
```

Begins with DEF, ends with FNEND. RETURN defines return value. Can pass and return arrays as well as simple and subscripted variables; one or two asterisks required in parentheses (*) or (*,*) after formal and actual parameter name to denote one or two dimensions; two asterisks required for string array. Other rules: local numeric variables, if any, must have type declared even if real; names may duplicate those used outside function body. Function calls permitted inside body, including recursive; second function definition not permitted. Cannot branch into or out of body. FOR loops and DO blocks must be completed inside body. Actual parameters may be altered by function if passed by reference (i.e., arrays, type-matched numeric variables, full-string variables) but not if passed by value (i.e., expressions, non-type-matched numeric variables, substrings). RETURN may follow THEN and ELSE in IF statement.

DEBUG COMMANDS

ABORT

Terminates program suspended by BREAK command or Y^C key and restores normal Interpreter mode.

BREAK

```
BREAK 100,150,200-210, PROG
BREAK
```

Causes program to suspend just prior to execution of specified statement or any statement within a range of specified statements. If PROG is specified, break occurs on each transfer from one program to another via CHAIN, INVOKE, or END. Interpreter identifies each break as it occurs:

```
*BREAK 100
*BREAK PROG6 940
*BREAK,INVOKE: PRTSECT
```

and prints command prompt. All debug commands are legal during break, plus limited set of others; see table on page 9. BREAK with no parameters immediately lists current breakpoints. GET, RUN with program name, and SCRATCH clear all breakpoints; see UNBREAK for clearing some or all breakpoints.

CALLS

Prints a list of all programs and functions called but not completed by a return. Reverse chronological order.

FILES

Prints a list of all files currently open in the executing program. Listed by internal file number (see FILES statement) and either: file name if file is open, asterisk if file number is reserved (not open), or internal file number preceded by # if file was opened by invoking program.

GO

```
GO
GO 250
```

Restarts program suspended by BREAK command at location where break occurred, or if a statement number is specified, at the specified statement. Cannot transfer into or out of a function. If break was caused by Y^C key, statement number cannot be specified.

RESUME

Same as GO.

SET

SET A=2.5, B$="ABC", FNC(M1)=5

Sets program variables to specified constant values. May set local variables in multiline user functions.

SHOW

SHOW A, B$, FNC(I,J,K), B(2,3), C(*), D$(*,*)

Prints values of items specified. Can specify local variables in multiline user functions (if active); also specific array items or entire arrays (one asterisk for numeric one-dimension, two for two-dimension or any string array). Array printed as in MAT PRINT, except undefined values noted by word UNDEFINED. Printed items are preceded by variable name; given only once for arrays.

TRACE

TRACE A, B$, FNC, FNC(I,J,K), B(2,3), D$(*,*)
TRACE 100, 150, 200-210, PROG
TRACE

Turns on tracing of selected variables, functions, local variables in functions, and statements, and of programs called. For variables: prints name and new value each time value changes, preceded by @ symbol and statement number where change occurred:

@ 80 A=65.5

For arrays (specified by one asterisk for numeric one-dimension, two for two-dimension or any string array): prints @ location, name, and new value for changed elements only, but not for MAT operations:

@ PROG9 120 D$(3)="ABCD"

For functions, prints function name preceded by @ when function is entered, and prints value when function is complete:

@ FNC
@ 270 FNC=5.5

For local variables of functions, only changes in variable value are traced, not execution of function. For statement tracing, program name (if applicable) and statement number are printed each time specified statement (or any statement within a specified range) is executed, preceded by *TRACE:

*TRACE 200

If PROG is specified, each invoke, return, and chain is printed, preceded by *TRACE:

*TRACE, INVOKE: PROG2
*TRACE, REVERT: PROG1
*TRACE, CHAIN: PROG3

TRACE with no parameters lists items currently being traced. GET, RUN with program name, and SCRATCH clear all tracing; UNTRACE clears selected or all tracing.

UNBREAK

UNBREAK 150, PROG
UNBREAK

With parameter list: deletes specified breakpoints set previously by BREAK. Without parameter list: deletes all current breakpoints.

UNTRACE

UNTRACE A, 200-210, FNC
UNTRACE

With parameter list: turns off tracing of specified items; turning off tracing of function does not affect tracing of local variables if they are being traced, and vice versa. Without parameter list: stops all tracing.

MPE COMMANDS

MPE/3000 Operating System commands can be entered after log-on (before entering BASIC) and during SYSTEM breaks (i.e., after using SYSTEM command or BREAK key). Exceptions: :JOB, :DATA, and :EOD are not valid in break mode.

:ABORT

Aborts the current program.

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

The commandfile is the source file (device) from which BASIC commands and statements are input; default is $STDINX. The inputfile is the source file which supplies data for INPUT, LINPUT, and ENTER statements; default is $STDINX. The listfile is the destination file for BASIC program listing and output; default is $STDLIST.

:BUILD filereference
[;DEV=device]
[;DISC=[filesize] [,[numextents] [initalloc]]]
[;REC=[recsize] [,[blockfactor] [[F / U / V] [,BINARY / ,ASCII]]]]
[;CCTL]
[;TEMP]
[;CODE=filecode]

Creates a new file.

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

Defines data from outside standard input stream. Cannot be read on $STDINX file. Acceptable for device recognition.

:EOD

Denotes end of data.

:FILE formaldesignator [=$NEWPASS | [=filereference] [,NEW]]
[;REC=[recsize][,[blockfactor][[F|U|V][;BINARY|;ASCII]]]]
[;CCTL|;NOCCTL]
[;NOBUF|;BUF[=numbuffers]]
[;EXC|;EAR|;SHR]
[;DEL|;SAVE|;TEMP]
[;DEV=device]
[;CODE=filecode]
[;DISC=[filesize][,[numextents][,initalloc]]]

For NEW FILES only; defines or redefines new file's characteristics.

:FILE formaldesignator [=$OLDPASS | [=filereference] [,OLD | ,OLDTEMP]]

[Other parameters same as for new files. The ";DISC=" parameter is valid only in conjunction with "$OLDPASS".]

For OLD FILES; defines or redefines old file's characteristics.

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

[Other parameters same as for new files.]

For SYSTEM-DEFINED files.

:FILE formaldesignator=$NULL

For NULL file.

:FILE *formaldesignator*=**formaldesignator*

For USER PRE-DEFINED files; redefines old file.

:JOB [jobname,] username[/upass].acctname[/apass] [,groupname[/gpass]] .
[;TERM=termtype]
[;TIME=cpulimit]
[;PRI=executionpriority]
[;INPRI=selectionpriority]
[;OUTCLASS=outputclass]

Initiates a batch job.

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

Lists descriptions of files.

:PTAPE [filename]

Reads a paper tape without X-OFF control.

:PURGE filereference[,TEMP]

Deletes a file from the system.

:RELEASE filereference

Releases security provisions of a file.

:RENAME oldfilereference,newfilereference[,TEMP]

Renames a file.

:REPORT

Displays total accounting information for a log-on group.

:RESET {@ | formaldesignator}

Resets a formal file designator, or all formal file *designators* (@).

:RESTORE tapefile[;[filesetlist] [[;KEEP] [;DEV=device] [;SHOW]]]

Restores a complete fileset, stored off-line.

:SAVE {[$OLDPASS,newfilereference] | [tempfilereference]}

:SECURE filereference

Restores suspended security provisions of a file.

:SHOWJOB
Displays job/session status.

:SPEED { [inspeed] ,outspeed / inspeed }
Changes input or output speed of terminal.

:STORE [filesetlist] ;tapefile[;SHOW]
Stores a set of files off-line.

:TELL { jsnumber / jsname } ;message
Transmits a message.

:TELLOP message
Transmits a message from the user to the computer operator.

MPE FILE OPERATIONS

BASIC BATCH JOBS

All commands and statements have the same format as for terminal mode sessions. Differences for jobs vs. sessions are:

1. Use :JOB/:EOJ rather than :HELLO/:BYE.
2. Do not place ">" at the beginning of statements and commands (or data). Sole exception: >EOD, which is required when commandfile and inputfile are the same, to separate input for different purposes.

READ CARDS IN SESSION

To read all data (for INPUT, LINPUT, and ENTER) from a card reader during a session, enter a file equation for the card reader, then back-reference the file in the inputfile parameter of the :BASIC command. (Comma and omitted third parameter accept defaults for commandfile and listfile.) Be sure card deck begins with a :DATA card (including passwords if any) and ends with an :EOD card.

```
:FILE CARDATA;DEV=CARD
:BASIC,*CARDATA
```

To read in and execute a program from a card reader, the BASIC/3000 XEQ command may be used.

```
:FILE CARDCMD;DEV=CARD
:BASIC
>XEQ CARDCMD
```


Or, back-reference the card reader file in the commandfile parameter of :BASIC.

```
:FILE CARDCMD;DEV=CARD
:BASIC *CARDCMD
```

In either case, the card deck must begin with a :DATA card, which must include the exact log-on identification as used to initiate the session, plus the file name, separated by a semicolon.

```
:DATA USER.ACCT;CARDCMD
```

READ DATA FROM DISC FILE

If all data for INPUT, LINPUT, and ENTER statements resides in an ASCII data file, specify that file in the inputfile parameter of :BASIC.

```
:BASIC,INDATA
```

LIST ON LINE PRINTER

To list all output (including program listing if requested) on a line printer, first enter a file equation for the line printer, then back-reference the file in the listfile parameter of the :BASIC command.

```
:FILE PRINTER;DEV=LP
:BASIC,,*PRINTER
```

Or, use the OUT= parameter of the BASIC RUN command.

```
:FILE PRINTER;DEV=LP
:BASIC
>GET PROG1
>RUN,OUT=PRINTER
```

Or, to direct only the program listing to the line printer, use the OUT= parameter of the BASIC LIST command.

```
:FILE PRINTER;DEV=LP
:BASIC
>GET PROG1
>LIST,OUT=PRINTER
```

READ FROM PAPER TAPE

To read data from paper tape without X-punching, or on KSR (non-ASR) terminals, the MPE command PTAPE may be used as well as the BASIC/3000 commands

SPOOL and TAPE. The advantage of PTAPE is that a permanent disc file is used.

```
:BUILD TAPEFILE; REC=-72,,V,ASCII
:PTAPE TAPEFILE
Turn on tape reader
Type Y^C when tape reading stops
:BASIC,TAPEFILE
```

Disc file called TAPEFILE is now inputfile.

STORE FILES ON MAGNETIC TAPE

To store a disc file on tape, enter file equation for tape file and back-reference in :STORE command:

```
:FILE T;DEV=TAPE
:STORE CARDFILE;*T;SHOW
```

Result:

```
FILES STORED = 1

  FILE     .GROUP     .ACCOUNT    LDN  ADDRESS

  CARDFILE.PUB        .ACCTNAME   2    %207631

FILES NOT STORED = 0
```

To store all disc files of all groups, use parameters @.@.acctname and back-reference the tape file:

```
:STORE @.@.ACCTNAME;*T
```

Result:

```
FILES STORED = 28

FILES NOT STORED = 0
```

DISC CARD FILE

To build a disc file for card images, using default file size of 1024 records:

```
:BUILD CARDFILE;REC=-80,,F,ASCII
```

FILE INFORMATION

To obtain description of particular disc file, including disc space used (detail 2):

```
:LISTF CARDFILE,2
```

Result:

```
ACCOUNT=  ACCTNAME    GROUP=  PUB

FILENAME  CODE  ------------LOGICAL RECORD-----------  ----SPACE----  ACC
                  SIZE  TYP        EOF      LIMIT R/B  SECTORS #X MX

CARDFILE          80B  FA           2        1023   3      43   1  8  ???
```


ASCII DECIMAL EQUIVALENTS

A	65	q	113	ETB	23
B	66	r	114	CAN	24
C	67	s	115	EM	25
D	68	t	116	SUB	26
E	69	u	117	ESC	27
F	70	v	118	FS	28
G	71	w	119	GS	29
H	72	x	120	RS	30
I	73	y	121	US	31
J	74	z	122	SPACE	32
K	75	0	48	!	33
L	76	1	49	"	34
M	77	2	50	#	35
N	78	3	51	$	36
O	79	4	52	%	37
P	80	5	53	&	38
Q	81	6	54	'	39
R	82	7	55	(	40
S	83	8	56	)	41
T	84	9	57	*	42
U	85	NUL	0	+	43
V	86	SOH	1	,	44
W	87	STX	2	-	45
X	88	ETX	3	.	46
Y	89	EOT	4	/	47
Z	90	ENQ	5	:	58
		ACK	6	;	59
a	97	BEL	7	<	60
b	98	BS	8	=	61
c	99	HT	9	>	62
d	100	LF	10	?	63
e	101	VT	11	@	64
f	102	FF	12	[	91
g	103	CR	13	\	92
h	104	SO	14	]	93
i	105	SI	15	^	94
j	106	DLE	16	_	95
k	107	DC1	17	`	96
l	108	DC2	18	{	123
m	109	DC3	19	|	124
n	110	DC4	20	}	125
o	111	NAK	21	~	126
p	112	SYN	22	DEL	127