Document Number:	DEC-11-LBACA-D-DN2
Document Name:	Change Notice to BASIC/RT11 Language Reference Manual
Date:	May, 1975
Maintainer:	Software Documentation

First Printing, May 1975

The HOW TO OBTAIN SOFTWARE INFORMATION page, located at the back of this document, explains the various services available to DIGITAL software users.

The postage prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

PREFACE

This document is a change notice to the BASIC/RT-11 Language Reference Manual (DEC-11-LBACA-D-D). Appendix H was omitted from the manual and is included in this change notice. Pages 5-27, 7-3, 8-8, 8-17, 9-3, E-3, F-2, F-5, I-6, J-2, and J-3 contained technical errors which have been fixed in this notice.

All changes are marked by bars in the outer margin and the date of the change in the lower left hand corner. Any previous change bars on a revised page have been deleted.

Insert APPENDIX H after APPENDIX G. The pages:

5-27
7-3/7-4
8-7/8-8
8-17
9-3/9-4
E-3/E-4
F-1/F-2
F-5
I-5/I-6
J-1/J-2
J-3/J-4

replace the equivalent pages in the BASIC/RT-11 Language Reference Manual.

The program in memory is now:

```
 10  DIM A(100)
 20  FOR I = 0 TO 100
 30  LET A(I) = SQR(I)
 40  NEXT I
 50  DEF FNS(I) = SQR (A(I))
 60  OPEN "LP:" AS FILE #1
100  PRINT #1: "FIRST OVERLAY"
110  FOR J = 1 TO 100
120  PRINT #1: FNS (J),
130  NEXT J
140  STOP
900  OVERLAY "OV1"
910  GO TO 100
```

Control now passes to statement 910, which is the first statement following statement 900 in the merged program.

Execution at statement 100 causes

```
"FIRST OVERLAY"
```

to be printed, followed by the fourth roots of the numbers from 0 to 100.

Finally, "STOP AT LINE 140" is output at the terminal.

An overlay statement executed in the immediate mode (without a line number) will act like an OLD command, except that the program currently in core is not scratched. Instead, the program lines in the specified file will be edited into the program, just as if they were typed in via the console.

A very useful application of this feature is when the BASIC programmer has a "library" of GOSUB subroutines to edit into his program. The procedure is as follows.

Type in the BASIC program as if there were subroutines at specific (high) statement numbers such as 1000, 2000, etc. Then SAVE the program. The next step is to resequence the required library routines using the BASIC program RESEQ (see Chapter 10) so that they begin at the correct statement numbers. Then read in the saved program again with the OLD command. Finally, edit in the subroutines with immediate mode OVERLAY statements such as

```
OVERLAY "SUB1"
OVERLAY "SUB2"
```

Finally, a REPLACE command will update the saved program.

NOTE

Execution of the OVERLAY statement may cause the data pointer to change. Any program employing both the OVERLAY and DATA statements should have a RESTORE statement executed after the OVERLAY statement. This will cause the data pointer to be at the start of the first DATA statement in the merged program. The OVERLAY statement changes the data pointer only in versions of BASIC/RT-11 before version 1B.

7.2 OLD COMMAND

The OLD command (OLD) erases the contents of the storage area (SCRATCH and CLEAR) and inputs the program via the specified device.

The form of the command is:

```
OLD "dev:filnam.ext"
```

If the file descriptor (dev:filnam.ext) is not specified as part of the OLD command, BASIC prints:

```
OLD FILE NAME--
```

and waits for the file description and the return key. Type the name of the file containing the BASIC program (do not enclose the filename in quotation marks). If a filename is not entered, BASIC assumes the name NONAME.

In the examples of OLD commands that follow, the computer printout is underlined

```
OLD
OLD FILE NAME--TEST1
```

clears user area and inputs program TEST1.BAS from Disk (DK).

```
OLD "DT1:PROG1"
```

clears user area and inputs program PROG1.BAS from DECtape unit 1.

```
OLD "PR:RESEQ"
```

clears user area and inputs the program RESEQ from the high speed paper tape reader.

7.3 LIST/LISTNH COMMANDS

The LIST command prints the specified lines of the user program currently in memory on the terminal. The program name, date and the BASIC version number are output as a header line for the lines being listed. The form of the LIST command is:

```
LIST statement no.-statement no.
```

Several variations of the LIST command can be used:

LIST statement no.	Lists only the specified line.
LIST-statement no.	Lists from the beginning of the program to and including the specified line.
LIST statement no.- LIST statement no.-END	Lists from the specified line to the end of the program.

LIST statement no.-statement no.
Lists the specified section of the program.

If no statement number is specified, the entire program is listed. If the statement number specified does not exist, the first line of the program is listed.

Typing LIST followed by the statement number causes the header line and the line specified to be listed. The LISTNH command also prints the lines currently in core but suppresses the header line.

Type CTRL/O (depress the CTRL key and type the O key) to suppress an undesired listing. BASIC returns to the READY message when command execution is complete.

The lines listed may differ slightly from those entered because:

1. Certain characters while acceptable to BASIC are stored in a standard manner when they appear outside of quotation marks.

Character typed	Character stored
]	)
[	(
=<	<=
=>	>=
><	<>

2. Literals are stored to 24 bits of accuracy. Those with more than 24 bits are truncated to 24 bits.

3. Although literal storage is 24 bits, output is truncated to 6 decimal digits.

4. Literals are output in standard BASIC format, regardless of how they were input, for example,

```
10 LET X=3.0+1.0000001
20 PRINT X-1E7
LIST
10 LET X=3+1
20 PRINT X-1.00000E+07
```

5. Spaces in the input program are ignored, except within strings and REM statements. The LIST command prints the program with spaces inserted to separate keywords and line numbers from numeric information. The listed program is therefore easier to read. In the case of an IF...GO TO statement, no space is typed before the GO TO keyword.

Examples:

```
LISTNH 100          lists line 100.
```

8.4 SYSTEM ROUTINES IN BASIC

The routines described below are all global symbols and are available to the user functions:

Routine Name (Global)	Call	Description
BOMB	TRAP 0 .ASCII 'MESSAGE' .EVEN	This routine stops execution of the BASIC program and types the message: ?MESSAGE AT LINE xxxx If the $LONGER option is specified, the '?' character is omitted. BASIC then types the READY message.
ERRPDL	JMP ERRPDL	Called when the stack pointer (SP) goes below the value in R4. Causes execution to halt and types out ?ETC AT LINE xxxxx. There are 20 extra "buffer" words on the stack. If the user routine will definitely not use more than this many words on the stack, the routine need not check for a stack overflow.
ERRSYN	JMP ERRSYN	Syntax error. Stops execution and prints out ?SYN AT LINE xxxxx.
ERRARG	JMP ERRARG	Argument error. Stops execution and prints out ?ARG AT LINE xxxxx.
EVAL	JSR PC,EVAL	Evaluate expression. R1 points to the start of the expression in the code. EVAL sets the carry bit as follows: carry = 0: The expression is numeric. The value of the expression is contained in the floating accumulator (FAC1 and FAC2). carry = 1: A string expression. If the string is non-null, the top of the stack is an indirect pointer to the string. (See section 8.6 for the format of string variables.) If the string is null, the top of the stack is the value 177777. In both cases, R1 is moved to point to the byte following the expression in the code. If it detects an error in the expression, EVAL branches to the appropriate error routine.

Routine Name (Global)	Call	Description

GETVAR	JSR PC,GETVAR	Address variable or array element. R2 must contain the address of the symbol table entry for the variable and R1 must point to the next byte beyond the second byte of the symbol table offset on call. GETVAR looks up and saves the address of the variable reference, so that a subsequent STOVAR or STOSVAR will store a value in the addressed variable. GETVAR destroys the FAC when addressing an array element; R1 is left unchanged unless the variable is subscripted, in which case R1 is advanced past the right parenthesis. To address the symbol table entry, precede the GETVAR call with the code:
		
		`MOVB (R1)+,R2 ;FIRST BYTE OF OFFSET`
		`BMI ESYN ;IF NEGATIVE, ERROR`
		`SWAB R2`
		`BISB (R1)+,R2 ;GET 2ND HALF OF OFFSET`
		`ADD (R5),R2 ;ADD BASE OF SYMBOL TABLE`

MSG	JSR R1,MSG .ASCIZ 'MESSAGE' .EVEN	Print message on console. Prints the ASCII characters specified after the JSR instruction up to the 0-byte. MSG prints only those characters specified in the calling sequence plus padding characters specific to the terminal in use. The calling program must insert a carriage return where required. MSG clears the CTRL/O condition.
STOVAR	JSR PC,STOVAR	Store numeric variable. Stores the FAC in the variable or array element last referenced by GETVAR. If it was a string variable, STOVAR stops execution of the program, and produces the ?NSM error message.
STOSVAR	JSR PC,STOSVAR	Store string variable. Stores the top of the stack in the variable or array element last referenced by GETVAR, and pops one word from the stack. If it was a numeric variable, STOSVAR stops execution of the program and produces the ?NSM error message.

The module defining the background routine should now be of the form

```
        .CSECT BKGMOD
        .GLOBL BKG
BKG:     ;START OF BACKGROUND ROUTINE
          .
          .
          .
         RTS PC
        .END
```

The BKGMOD object module should be linked with FTBL, PERVEC, RTINT, the appropriate LPS and GT object modules, and the BASIC object modules.

Table 9-1 (Cont.)
BASIC Error Messages

Abbreviation	Message	Explanation
?NBF	NEXT BEFORE FOR AT LINE xxxxx	The NEXT statement corresponding to a FOR statement precedes the FOR statement.
?NER	NOT ENOUGH ROOM	There is not enough room on the selected device for the specified number of output blocks.
?NPR	NO PROGRAM	The RUN command has been specified, but no program has been typed in.
?NSM	NUMBERS AND STRINGS MIXED AT LINE xxxxx	String and numeric variables may not appear in the same expression, nor may they be set = to each other; for example, A$=2.
?OOD	OUT OF DATA AT LINE xxxxx	The data list was exhausted and a READ requested additional data.
?OVF	OVERFLOW AT LINE xxxxx	The result of a computation is too large for the computer to handle.
?PTB	PROGRAM TOO BIG	The line just entered caused the program to exceed the user code area.
?RBG	RETURN BEFORE GOSUB AT LINE xxxxx	A RETURN was encountered before execution of a GOSUB statement.
?RPL	USE REPLACE	File already exists. Use REPLACE command.
?SOB	SUBSCRIPT OUT OF BOUNDS AT LINE xxxxx	The subscript computed is greater than 32,767 or is outside the bounds defined in the DIM statement.
?SSO	STRING STORAGE OVERFLOW AT LINE xxxxx	There is not enough core available to store all the strings used in the program.
?STL	STRING TOO LONG AT LINE xxxxx	The maximum length of a string in a BASIC statement is 255 characters.
?SYN	SYNTAX ERROR AT LINE xxxxx	The program has encountered an unrecognizable statement. Common examples of syntax errors are misspelled commands and unmatched parentheses, and other typographical errors.

(Continued on next page)

Table 9-1 (Cont.)

BASIC Error Messages

Abbreviation	Message	Explanation
?TLT	LINE TOO LONG TO TRANSLATE	Lines are translated as entered and the line just entered exceeds the area available for translation.
?UFN	UNDEFINED FUNCTION AT LINE xxxxx	The function called was not defined by the program or was not loaded with BASIC.
?ULN	UNDEFINED LINE NUMBER AT LINE xxxxx	The line number specified in an IF, GO TO or GOSUB statement does not exist anywhere in the program.
?WLO	WRITE LOCKOUT	Tried to write on a sequential or virtual file opened for input only.
?↑ER	↑ ERROR AT LINE xxxxx	The program tried to compute the value A↑B, where A is less than 0 and B is not an integer. This produces a complex number which is not represented in BASIC.

When the message ?DNR AT LINE xxxxx is printed because the device referenced is not on-line, turn the device on and issue a GO TO xxxxx statement. Execution of the program resumes at the line (xxxxx) specified. This message may also indicate that a program file does not contain any legal BASIC program lines.

When the message ?OOD AT LINE xxxxx is printed because the file referenced by an INPUT#1 statement is not ready, prepare the file and issue a GO TO statement to resume execution.

Function Errors

The following errors can occur when a function is called improperly.

?ARG	The argument used is the wrong type. For example, the argument was numeric and the function expected a string expression.
?SYN	The wrong number of arguments was used in a function, or the wrong character was used to separate them. For example, PRINT SIN(X,Y) will produce a syntax error.

In addition, the functions give the errors listed below.

FNa(...)	?UFN	The function a has not been defined (function cannot be defined by an immediate mode statement).
RND or RND(X)		No errors
SIN(X)		No errors

Abbreviation	Message	Explanation
?OOD	OUT OF DATA AT LINE xxxxx	The data list was exhausted and a READ requested additional data.
?OVF	OVERFLOW AT LINE xxxxx	The result of a computation is too large for the computer to handle.
?PTB	PROGRAM TOO BIG	The line just entered caused the program to exceed the user code area.
?RBG	RETURN BEFORE GOSUB AT LINE xxxxx	A RETURN was encountered before execution of a GOSUB statement.
?RPL	USE REPLACE	File already exists. Use REPLACE command.
?SOB	SUBSCRIPT OUT OF BOUNDS AT LINE xxxxx	The subscript computed is greater than 32,767 or is outside the bounds defined in the DIM statement.
?SSO	STRING STORAGE OVERFLOW AT LINE xxxxx	There is not enough core available to store all the strings used in the program.
?STL	STRING TOO LONG AT LINE xxxxx	The maximum length of a string in a BASIC statement is 255 characters.
?SYN	SYNTAX ERROR AT LINE xxxxx	The program has encountered an unrecognizable statement. Common examples of syntax errors are misspelled commands and unmatched parentheses, and other typographical errors.
?TLT	LINE TOO LONG TO TRANSLATE	Lines are translated as entered and the line just entered exceeds the area available for translation.
?UFN	UNDEFINED FUNCTION AT LINE xxxxx	The function called was not defined by the program or was not loaded with BASIC.
?ULN	UNDEFINED LINE NUMBER AT LINE xxxxx	The line number specified in an IF, GO TO or GOSUB statement does not exist anywhere in the program.
?WLO	WRITE LOCKOUT AT LINE xxxxx	Tried to open a read-only device for output, or tried to write on a sequential or virtual file opened for input only.

Abbreviation	Message	Explanation
?↑ER	↑ERROR AT LINE xxxxx	The program tried to compute the value A↑B, where A is less than 0 and B is not an integer. This produces a complex number which is not represented in BASIC.

Function Errors

The following errors can occur when a function is called improperly.

?ARG	The argument used is the wrong type. For example, the argument was numeric and the function expected a string expression.
?SYN	The wrong number of arguments was used in a function, or the wrong character was used to separate them. For example, PRINT SIN(X,Y) produces a syntax error.

In addition, the functions give the errors listed below.

FNa(...)	?UFN	The function a has not been defined (function cannot be defined by an immediate mode statement).
RND or RND(X)		No errors
SIN(X)		No errors
COS(X)		No errors
SQR(X)	?ARG	X is negative
ATN(X)		No errors
EXP(X)	?↑ER	X is greater than 87
LOG(X)	?ARG	X is negative or 0
ABS(X)		No errors
INT(X)		No errors
SGN(X)		No errors
TAB(X)	?ARG	X is not in the range $0 \leq x < 256$
LEN(A$)		No errors
ASC(A$)	?ARG	A$ is not a string of length 1
CHR$(X)	?ARG	X is not in the range $0 \leq x < 256$

APPENDIX F

ASSEMBLING AND LINKING BASIC

F.1 ASSEMBLING BASIC/RT11

The source program of BASIC/RT11 consists of three source files: A 16K system is required to assemble BASIC.

```
BASICL.MAC
BASICH.MAC
FPMP.MAC
```

It is necessary to create the files BASICR, BASICE, and BASICX which consist of only one line of code each. They specify the conditionals necessary to assemble BASICL into the three object modules BASICR.OBJ, BASICE.OBJ and BASICX.OBJ.

They are created using the EDIT program, as follows:

```
($) Represents the Altmode key
.R EDIT
*EWBASICR.MAC ($)($)
*IBASICR=1
($) EX ($)($)

.R EDIT
*EWBASICE.MAC ($)($)
*IBASICE=1
($) EX ($)($)

.R EDIT
*EWBASICX.MAC ($)($)
*IBASICX=1
($) EX ($)($)
```

If any other options are desired, include the conditionals for them in these files. For example:

```
$NOSTR=1        ;NO STRINGS
$LONGER=1       ;LONG ERROR MESSAGES
$NOVF=1         ;NO VIRTUAL MEMORY FILES
$NOPOW=1        ;NO POWER-FAIL OPTION
$STKSZ=n        ;PROGRAM STACK SIZE
                ;IN BYTES (DEFAULT IS
                ;200 (OCTAL) BYTES
```

If BASIC is to run on an 8K system, the $NOSTR conditional must be specified.

For example, to create a BASIC with no strings, no virtual memory files, and a stack size of 300 (octal) the BASICR, BASICE, and BASICX files should be created using the EDIT program, as follows

```
.R EDIT
*EWBASICR.MAC ($)($)
*IBASICR=1
$NOSTR=1
$NOVF=1
$STKSZ=300
($) EX ($)($)
```

```
.R EDIT
*EWBASICE.MAC ($)($)
*IBASICE=1
$NOSTR=1
$NOVF=1
$STKSZ=300
($) EX ($)($)

.R EDIT
*EWBASICX.MAC ($)($)
*IBASICX=1
$NOSTR=1
$NOVF=1
$STKSZ=300
($) EX ($)($)
```

($) represents the Altmode key.

To assemble Basic, type the following as input to the MACRO Assembler:

```
*BASICR=BASICR,BASICL
*BASICE=BASICE,BASICL
*BASICX=BASICX,BASICL
*BASICH=BASICH
*FPMP=FPMP
```

This produces the five object modules

BASICR	BASIC Root section
BASICE	BASIC Edit overlay
BASICX	BASIC EXecution overlay
FPMP	Floating Point Math Package
BASICH	BASIC High section, with once-only code and optional functions

F.1.1 Floating Point Math Package

Assembly of the FPMP source file produces a "standard" FPMP for BASIC, which runs on any PDP-11, but will not make use of special arithmetic hardware. All of the routines needed for the full complement of BASIC arithmetic functions are included. A non-standard FPMP may be specified, as outlined in the table below:

FPMP Assembly Parameters

Parameter	Default Value	Description
MIN	undefined	Define to eliminate code for BASIC functions SIN, COS, SQR, and ATN. When linked, the functions are listed as "undefined references". However, when executed by a BASIC program, they produce a ?UFN (UNDEFINED FUNCTION) error.

To link BASIC with the user functions in a non-overlay system, type this command string to the Linker:

```
*BASIC=BASICR,FPMP,BASICE,BASICX/B:4ØØ/C

*FUN1,FUN2[,GETARG],BASICH
```

GETARG is the general argument interface module listed in Appendix H. In an overlay system, there are two possible ways in which to link BASIC with the user functions.

If the user function routines contain no data which must be preserved from one function call to the next, that is, if the code for the routines may be refreshed at the beginning of each function call, then the routines may be incorporated into the execution overlay by using this LINK command string:

```
*BASIC,BASIC=BASICR,FPMP,FUN1/T/B:4ØØ/C
TRANSFER ADDRESS =
GO
*BASICE/O:1/C
*BASICX,FUN2[,GETARG]/O:1/C
*BASICH/O:2
```

In this case, the function routines (in the module FUN2) occupy space in the first overlay segment which is normally unused, since the Edit overlay segment (BASICE) is about 250 words longer in the 8K no-string system than the Execution overlay segment (BASICX). These first 250 words of storage are "free" in this case.

In the case where FUN2 may not be read in anew whenever it is used, type this command string to the Linker:

```
*BASIC=BASICR,FPMP,FUN1,FUN2/T/B:4ØØ/C
TRANSFER ADDRESS =
GO
*BASICE/O:1/C
*BASICX[,GETARG]/O:1/C
*BASICH/O:2
```

There are three additional object modules (FPMP.FPU, FPMP.EAE, FPMP.EIS) which allow BASIC/RT11 to be linked for special arithmetic hardware.

Processor	Replace FPMP.OBJ With
EAE hardware	FPMP.EAE
PDP-11/40 extended processor or PDP-11/45 processor	FPMP.EIS
PDP-11/45 FPU hardware	FPMP.FPU

APPENDIX H

GETARG, STORE, SSTORE LISTING

```
; GETARG, STORE, SSTORE :  SUBROUTINES FOR
; LINKAGE OF ASSEMBLER SUBROUTINES TO BASIC
;
        .TITLE  GETARG  29-AUG-73
        .GLOBL  GETARG, STORE
        .GLOBL  EVAL, GETVAR, ERRARG, ERRSYN
        .GLOBL  .LPAR, .COMMA, .RPAR, .EOL
        .GLOBL  STOVAR, .SQUOT, .DQUOT
        .IFNDF  $NOSTR
        .GLOBL  SSTORE, STOSVAR
        .ENDC   ;$NOSTR
        .CSECT  GET
;
;$NOSTR =        1       ;DELETE ';' TO ASSEMBLE FOR
;                        ;BASIC WITH NO STRINGS
R0=%0
R1=%1
R2=%2
R3=%3
R4=%4
R5=%5
SP=%6
PC=%7
NVAL=4
        .IFDF   $NOSTR
NVAL=3
        .ENDC   ;$NOSTR
.TEXT=377
FAC1=40
FAC2=42
VARSAV=22
```

```
;-------------------------------------------------------------
; SUBROUTINE 'GETARG'   CALLED BY MOV  #TABLE,R0
;                                  JSR  PC,GETARG
;                                  .BYTE N1,N2,...,0
;                                  .EVEN
;                       WHERE TABLE IS THE ADDRESS OF A
;                       TABLE TO HOLD THE ARG REFERENCES.
;                       N1,N2,ETC. INDICATE THE ARG TYPES:
;                       1       INPUT NUMERIC EXPRESSION. 2
;                               (THE EXPRESSION VALUE) ARE
;                               STORED IN TABLE.
;                       2       OUTPUT NUMERIC VARIABLE. 3 WORDS
;                               ARE STORED IN TABLE.
;                       STRING VERSION ONLY:
;                       3       INPUT STRING EXPRESSION. NO WORDS
;                               ARE STORED IN TABLE. THE STRING
;                               POINTER IS ON THE STACK.
;                       4       OUTPUT STRING VARIABLE. 3 WORDS
;                               ARE STORED IN TABLE.
;                       NO STRING VERSION:
;                       3       INPUT STRING LITERAL. 2 WORDS
;                               ARE STORED IN TABLE. WORD 1 CON-
;                               TAINS THE START OF THE ASCII STRING.
;                               WORD 2 CONTAINS THE LENGTH OF THE
;                               STRING IN BYTES.
;                       CHECKS THE SYNTAX OF THE CALLING
;                       STATEMENT AND FINDS THE REQUESTED
;                       ARGUMENT REFERENCES, STORING THEM
;                       CONSECUTIVELY IN TABLE.
GETARG: MOV     (SP)+,R3        ;ADDR OF CALL IN R3
        MOVB    (R3)+,R2        ;GET 1ST BYTE IN R2
        BLE     GETX            ;NO ARGS, EXIT
        CMPB    (R1)+,#.LPAR    ;CHECK STARTING '('
        BNE     GETERS          ;NO, SYNTAX ERROR
        BR      GET2            ;ENTER LOOP
GET1:   CMPB    (R1)+,#.COMMA   ;CHECK ',' BETWEEN ARGS
        BNE     GETERS          ;NO, SYNTAX ERROR
GET2:   CMP     R2,#NVAL        ;CHECK VALID BYTE
        BHI     GETERA
        ASL     R2
        MOV     R0,R0S          ;SAVE REGS
        MOV     R3,R3S
        MOV     BRTAB-2(R2),PC  ;BRANCH TO ROUTINE
```

```
; NUMERIC EXPRESSION
NUMEXP: JSR     PC,EVAL         ;EVALUATE!
        BCS     GETERA          ;STRING IS BAD
        MOV     R0S,R0          ;RESTORE TABLE POINTER
        MOV     FAC1(R5),(R0)+
        MOV     FAC2(R5),(R0)+  ;SAVE VALUE
        BR      NXTARG
; STRING EXPRESSION
STREXP:
        .IFNDF  $NOSTR
        JSR     PC,EVAL         ;EVALUATE!
        BCC     GETERA          ;NUMERIC IS BAD
        MOV     R0S,R0          ;RESTORE TABLE POINTER
        BR      NXTARG
        .ENDC   ;$NOSTR
        .IFDF   $NOSTR
        MOVB    (R1)+,-(SP)     ;LOOK FOR STRING LITERAL
        CMPB    (SP),#.SQUOT    ;CHECK QUOTE CHAR.
        BEQ     STR1
        CMPB    (SP),#.DQUOT
        BNE     GETERS
STR1:   CMPB    (R1)+,#.TEXT    ;CHECK .TEXT TOKEN NEXT
        BNE     GETERS
        MOV     R0S,R0          ;RESTORE TABLE POINTER
        MOV     R1,(R0)+        ;SAVE STRING ADDRESS IN TABLE
        CLR     R2              ;NOW FIND LENGTH
STR2:   TSTB    (R1)+           ;END OF STRING IS BYTE 00
        BEQ     STR3
        INC     R2              ;COUNT
        BR      STR2
STR3:   MOV     R2,(R0)+        ;SAVE LENGTH IN TABLE
        CMPB    (SP)+,(R1)+     ;CHECK MATCHING CLOSE QUOTE
        BNE     GETERS
        BR      NXTARG
        .ENDC   ;$NOSTR
; NUMERIC TARGET VARIABLE
NUMVAR: CLR     -(SP)           ;REMEMBER IT'S NUMERIC
        .IFNDF  $NOSTR
        BR      VAR1
; STRING TARGET VARIABLE
STRVAR: MOV     R2,-(SP)        ;REMEMBER IT'S STRING
        .ENDC   ;$NOSTR
VAR1:   MOVB    (R1)+,R2        ;GET SYMTAB REF IN R2
        BMI     GETERS
        SWAB    R2
        BISB    (R1)+,R2
        ADD     (R5),R2
        JSR     PC,GETVAR       ;ADDRESS VARIABLE
        MOV     R0S,R0          ;RESTORE TABLE POINTER
        MOV     R5,R2           ;ADDRESS VARSAV
        ADD     #VARSAV,R2
        MOV     (R2),R3         ;SAVE A COPY
        MOV     (R2)+,(R0)+     ;MOVE 3 WORDS INTO TABLE
        MOV     (R2)+,(R0)+
        MOV     (R2),(R0)+
        TST     (SP)+           ;STRING OR NUM
        BNE     VAR2
        CMP     (R3),#-1        ;NUMERIC, CHECK TYPE AGREES
        BEQ     GETERA
        BR      NXTARG
VAR2:   CMP     (R3),#-1
        BNE     GETERA
```

```
; GO TO NEXT ARGUMENT
NXTARG: MOV     R3S,R3
        MOVB    (R3)+,R2        ;GET NEXT BYTE IN R2
        BGT     GET1            ;LOOP TILL BYTE IS 0
        CMPB    (R1)+,#.RPAR    ;CHECK CLOSING ')'
        BNE     GETERS
GETX:   CMPB    (R1)+,#.EOL     ;AND END-LINE TOKEN
        BNE     GETERS
        INC     R3              ;MAKE SURE R3 IS EVEN
        ASR     R3
        ASL     R3
        JMP     (R3)
R0S:    .WORD   0
R3S:    .WORD   0
GETERA: JMP     ERRARG
GETERS: JMP     ERRSYN
BRTAB:  .WORD   NUMEXP
        .WORD   NUMVAR
        .WORD   STREXP
        .IFNDF  $NOSTR
        .WORD   STRVAR
        .ENDC   ;$NOSTR
```

```
;-------------------------------------------------------
; SUBROUTINE 'STORE'    CALLED BY JSR PC,STORE
;                       R0 POINTS TO 3-WORD ARG REFERENCE
;                       SET UP BY GETVAR
;                       SAVES THE VALUE OF THE FAC
;                       IN THE SPECIFIED NUMERIC VARIABLE
STORE:  MOV     R5,R2           ;ADDRESS VARSAV
        ADD     #VARSAV,R2
        MOV     (R0)+,(R2)+     ;MOVE FROM TABLE TO USER AREA
        MOV     (R0)+,(R2)+
        MOV     (R0),(R2)
        JSR     PC,STOVAR       ;STORE IT
        RTS     PC
;
        .IFNDF  $NOSTR
;-------------------------------------------------------
; SUBROUTINE 'SSTORE'   CALLED BY JSR PC,SSTORE
;                       R0 POINTS TO 3-WORD ARG REFERENCE
;                       SET UP BY GETVAR
;                       STRING POINTER IS AT THE TOP OF STK
;                       SAVES THE STRING AT TOP OF STK
;                       IN THE SPECIFIED STRING VARIABLE
SSTORE: MOV     R5,R2
        ADD     #VARSAV,R2      ;ADDRESS VARSAV
        MOV     (R0)+,(R2)+     ;MOVE FROM TBL TO USER AREA
        MOV     (R0)+,(R2)+
        MOV     (R0),(R2)
        MOV     (SP),R3         ;SWITCH RETURN & STRING PTR
        MOV     2(SP),(SP)
        MOV     R3,2(SP)
        JSR     PC,STOSVAR      ;STORE STRING
        RTS     PC              ;RETURN
        .ENDC   ;$NOSTR
        .END
```

I.3.2 "ACC"(BUF)

Access entire buffer BUF. This command resets all buffer pointers of the array BUF to allow full access to it by the RDB and PUTD commands. The PUTD pointer is placed at the end of the array and the RDB pointer is placed at the beginning.

Example:

Allow full access to the array H and the array A(11).

```
 10 DIM A(25),H(20)
 20 CALL "USE"(A(1),A(11),A(31),A,H)
 30 ....
 40 ....
  ..
  ..
  ..
100 CALL "ACC"(H)
110 CALL "ACC"(A(11))
120 ....
130 ....
  ..
  ..
  ..
```

I.3.3 "RDB"(BUF,var)

Return the next data point from the specified buffer. Returns values of 65535>=var>=0 for good data. Bad data (defined as overrun) is returned as a minus one. If no data exists yet, a minus 2 will be returned.

When the referenced buffer refers to analog sampling (RTS function), the values returned are in the range 4095>=var>=0.

When the referenced buffer refers to a clocked histogram sampling (HIST function), the values returned are in the range 65535>=var>=0. These values are either the number of ticks accumulated or the number remaining depending on the clock mode.

When the referenced buffer refers to a Digital I/O operation (DRS function), a value between 65535>=var>=0 is returned from the next position in the specified buffer.

Example:

Assume that the array X has 100 data values previously entered by an RTS command. Print out the data making sure that data overrun did not occur and that 100 data points were indeed taken.

```
  ..
  ..
100 FOR I=1 TO 100
110 CALL "RDB"(X,Z)
120 IF Z >= 0 GO TO 160
130 IF Z =-2 GO TO 180
140 PRINT "BAD DATA AT EVENT";I
150 GO TO 190
160 PRINT Z
```

```
170 GO TO 190
180 PRINT "NO DATA AT EVENT";I
190 NEXT I
 ..
 ..
```

I.4 MODULE 1 (A/D CONVERSION AND NUMERIC READOUTS)

I.4.1 "ADC"(chan,var)

Initiate an A/D conversion from the specified channel (0<=chan<=15), wait for it to complete, and return the conversion as a floating point result in "var" (0<=var<=4095). The A/D cannot be currently involved in a Real-Time Sampling (RTS) operation.

Example:

Sample the A/D from channels 4 and 5 and save the results in the arrays A4 and A5 respectively. Assume 100 samples are to be taken.

```
10 DIM A4(100),A5(100)
20 FOR I=1 TO 100
30 CALL "ADC"(4,A4(I))
40 CALL "ADC"(5,A5(I))
50 NEXT I
```

I.4.2 "RTS"(BUF,sc,nsc,npts,mode)

Perform real time buffered/clocked sampling of the A/D. The A/D can be enabled in a variety of options depending on the mode specified. The normal mode of operation (mode=0) causes the A/D to sample whenever Schmitt trigger 1 fires. A mode of 2 causes the A/D to sample whenever the clock overflows. To enable other options, merely add their code number to the mode. The following list describes options available (all options are normally disabled):

Code	Option
+1	Enable burst mode (used only with DMA)
+2	Enable clock, disable Schmitt trigger 1
+4	Enable dual sample and hold
+8	Enable DMA

The A/D will be started by a clock overflow or the firing of Schmitt trigger 1. Pointers are used to determine if good data exists in the buffer arrays or if data wraparound occurs. Since data is stored in circular buffers (excluding DMA operations), pointers are used to ensure that the incoming data rate does not exceed the removal rate. Data returned as minus 2 (-2) indicates that data overrun occurred. The buffer pointers are reset initially before the sampling operation begins.

A/D channels are sampled on every clock overflow or firing of Schmitt trigger 1 with the result stored in consecutive data cells. Data is stored in a format identical to that read from the A/D. When a clock

APPENDIX J

GT GRAPHICS SUPPORT

J.1 INTRODUCTION

BASIC is provided with GT Graphics support for the GT44 and GT40 Display Processors. The support consists of a collection of routines accessible by the CALL statement. These routines allow BASIC programs to have complete control of the display processor.

Points, vectors, text, and graph data may all be combined through simple CALL statements. The screen may easily be scaled to any coordinates. Portions of the display may be controlled independently through use of the subpicture feature. Special graphic routines allow the display of an entire array of data by one call statement. The area of core that is allocated to the display buffer may be dynamically controlled.

When operating in the RT-11 environment, any display may be saved as a file on a mass storage device with the exception of graph arrays. This file may later be restored which will cause the original display to appear on the screen without the BASIC program originally needed to create it.

Support is provided for a real-time clock. The graphics support package will link with and support the Laboratory Peripheral System support that is also provided with BASIC.

The hardware required for use of the BASIC GT Graphics support is a GT40 or GT44 processor, a VT11 display screen, 16K or more of core memory, and a user's terminal. In addition to the peripheral input/output device needed to support the BASIC system (disk, DECtape, cassette, or paper tape), the calls to TIME and TIMR require a real-time clock. The core required for the Graphics support itself is approximately 2.5K in a core resident form and 2.1K in an overlay form.

The documentation for BASIC with Graphics support is provided in two parts the BASIC Manual (BASIC/RT11 Language Reference Manual) and this appendix. All information concerning BASIC arithmetic, strings, operations, functions, statements, and commands may be found in the BASIC Manual. This appendix describes the use of the BASIC calls to the GT Graphic routines. A general description of the CALL statement may be found in section 8.1 of the BASIC/RT11 Language Reference Manual.

The GT Support is supplied in the BASIC kit in the following files:

GTB.OBJ	Main GT object module
GTC.OBJ	GT object module that may be linked in an overlay (otherwise it is linked in core)
PERVEC.MAC	Vector definition source file

FTBL.MAC	Function table
BASINT.MAC	Interface Module
RTINT.MAC	Interface Module for BASIC/RT11 V01
PTSINT.MAC	Interface Module for BASIC/PTS V01
PERPAR.MAC	Parameter file
GTNLPS.OBJ	Module linked with GT when LPS support is not also linked

For instructions to build a load module of BASIC with GT support see Section J.3. Software for BASIC/RT11 with GT support that is provided on DECtape, cassette and DECpack disk also contains two running versions of BASIC:

BASGT.SAV	BASIC with GT support
BGTLPS.SAV	BASIC with GT and LPS support

BASGT.SAV is a non-overlaying version of BASIC with GT support. BGTLPS.SAV is a non-overlaying version of BASIC with GT and LPS support. BASGT.SAV is loaded by the following RT-11 monitor command:

```
.R BASGT
```

To load a version of BASIC with GT and LPS support the following command should be given:

```
.R BGTLPS
```

At this point the standard BASIC initial dialogue will occur. See Chapter 1 of the BASIC/RT11 Language Reference Manual for a description of the initial dialogue. As part of the initial dialogue BASIC will print:

```
USER FNS LOADED
```

This message will be printed whenever BASIC has had GT support linked with it. BASIC will terminate the initial dialogue by printing:

```
READY
```

NOTE

BASIC with GT support should not be run by the RT-11 monitor after GTON, a program supplied with RT-11 (version 1 only), has been run. GTON causes RT-11 to print all information on the graphic display screen and any attempt by BASIC with GT support to use the display screen causes the computer to halt. If this happens, the monitor must be rebooted. To avoid this, when GTON has been run, do not run BASIC until the monitor has been rebooted by either a hardware bootstrap or the PIP reboot command. See Section 4.13 of

the RT-11 System Reference Manual for a description of the PIP command. BASIC/RT-11 V01B with GT support is compatible with the RT-11 (version 2 or later) GT ON monitor command.

J.1.1 Documentation Conventions

The following chart describes the documentation conventions used in the description of the GT calls in this Appendix.

CONVENTION	MEANING
Square Brackets [ ]	Optional arguments are enclosed.
Lower case letter or lower case letter followed by a digit (a,b,x0,y1)	Value to be supplied by user -- may be any valid arithmetic expression.
Lower case letter followed by a dollar sign, (a$,x$)	String to be supplied by user may be string constant (enclosed in quotes) or variable (A$).
Upper case letter (A,B,X,Y)	Numeric variable whose value will be determined by call or an array name.
Y axis	The vertical axis
X axis	The horizontal axis

J.2 DISPLAY PROCESSOR CONTROL ROUTINES - CALL SUMMARY

BASIC programs can control the GT44 display processor by the use of the twenty-nine routines that are supplied with GT support for BASIC. A complete description of the BASIC call statement may be found in section 8.1 of the BASIC/RT11 Language Reference Manual.

The format of the CALL statement is:

```
CALL "name" (argument list)
        or
LET A$="name"
CALL (A$) (argument list)
```

The following chart summarizes the names, argument lists, and effects of the graphic calls supplied with the GT graphic support.

Call	Argument List	Effect
AGET	(A(i), Z)	Unscales element i of the graphic array A and stores in Z.

Call	Argument List	Effect
APNT	(x,y [,l,i,f,t])	Positions beam at point represented by (x,y) after scaling. Optional changing of l,i,f, and t parameters. l is light pen sensitivity, i is intensity, f is flash, and t is line type.
APUT	(A (i), b)	Assigns element i of the graphic array A the scaled value of b. Dynamically changes display of array A.
DCNT		Restores to the screen display stopped by call to DSTP.
DFIX (n)		Eliminates old display buffer if it exists and creates a display buffer of n words. Closes all open BASIC files.
DON (t)		Turns on subpicture with tag t that had been turned off with a call to OFF.
DSAV	[("[dev:]filname[.ext]")]	Compacts the display file by eliminating references to erased subpictures and graphics arrays and if a file is specified creates a copy of the graphic display on a file on DECtape or disk. Display may then be restored to the screen at any time by a call to RSTR. DK: is the default device. DPY is the default extension.
DSTP		Stops display of the entire display buffer. Display may be restored by a call to DCNT.
ERAS	[(t)]	Erases subpicture with the tag. If t is not specified this call erases the tracking object created by a call to TRAK.
ESUB		Terminates subpicture created by a call to SUBP (with one argument).
FIGR	(A[,l,i,f,t])	Creates vectors from array A to form figure. See section J.2.8 concerning the cautions required when using graphic array calls. Optional changing of l,i,f, and t parameters.

HOW TO OBTAIN SOFTWARE INFORMATION

SOFTWARE NEWSLETTERS, MAILING LIST

The Software Communications Group, located at corporate headquarters in Maynard, publishes software newsletters for the various DIGITAL products. Newsletters are published monthly, and keep the user informed about customer software problems and solutions, new software products, documentation corrections, as well as programming notes and techniques.

There are two similar levels of service:

- The Software Dispatch
- The Digital Software News

The Software Dispatch is part of the Software Maintenance Service. This service applies to the following software products:

PDP-9/15
RSX-11D
DOS/BATCH
RSTS-E
DECsystem-10

A Digital Software News for the PDP-11 and a Digital Software News for the PDP-8/12 are available to any customer who has purchased PDP-11 or PDP-8/12 software.

A collection of existing problems and solutions for a given software system is published periodically. A customer receives this publication with his initial software kit with the delivery of his system. This collection would be either a Software Dispatch Review or Software Performance Summary depending on the system ordered.

A mailing list of users who receive software newsletters is also maintained by Software Communications. Users must sign-up for the newsletter they desire. This can be done by either completing the form supplied with the Review or Summary or by writing to:

Software Communications
P.O. Box F
Maynard, Massachusetts 01754

SOFTWARE PROBLEMS

Questions or problems relating to DIGITAL's software should be reported as follows:

North and South American Submitters:

Upon completion of Software Performance Report (SPR) form remove last copy and send remainder to:

Software Communications
P.O. Box F
Maynard, Massachusetts 01754

The acknowledgement copy will be returned along with a blank SPR form upon receipt. The acknowledgement will contain a DIGITAL assigned SPR number. The SPR number or the preprinted number should be referenced in any future correspondence. Additional SPR forms may be obtained from the above address.

All International Submitters:

Upon completion of the SPR form, reserve the last copy and send the remainder to the SPR Center in the nearest DIGITAL office. SPR forms are also available from our SPR Centers.

PROGRAMS AND MANUALS

Software and manuals should be ordered by title and order number. In the United States, send orders to the nearest distribution center.

Digital Equipment Corporation
Software Distribution Center
146 Main Street
Maynard, Massachusetts 01754

Digital Equipment Corporation
Software Distribution Center
1400 Terra Bella
Mountain View, California 94043

Outside of the United States, orders should be directed to the nearest Digital Field Sales Office or representative.

USERS SOCIETY

DECUS, Digital Equipment Computers Users Society, maintains a user exchange center for user-written programs and technical application information. The Library contains approximately 1,900 programs for all DIGITAL computer lines. Executive routines, editors, debuggers, special functions, games, maintenance and various other classes of programs are available.

DECUS Program Library Catalogs are routinely updated and contain lists and abstracts of all programs according to computer line:

- PDP-8, FOCAL-8, BASIC-8, PDP-12
- PDP-7/9, 9, 15
- PDP-11, RSTS-11
- PDP-6/10, 10

Forms and information on acquiring and submitting programs to the DECUS Library may be obtained from the DECUS office.

In addition to the catalogs, DECUS also publishes the following:

DECUSCOPE	-The Society's technical newsletter, published bi-monthly, aimed at facilitating the interchange of technical information among users of DIGITAL computers and at disseminating news items concerning the Society. Circulation reached 19,000 in May, 1974.
PROCEEDINGS OF THE DIGITAL EQUIPMENT USERS SOCIETY	-Contains technical papers presented at DECUS Symposia held twice a year in the United States, once a year in Europe, Australia, and Canada.
MINUTES OF THE DECsystem-10 SESSIONS	-A report of the DECsystem-10 sessions held at the two United States DECUS Symposia.
COPY-N-Mail	-A monthly mailed communique among DECsystem-10 users.
LUG/SIG	-Mailing of Local User Group (LUG) and Special Interest Group (SIG) communique, aimed at providing closer communication among users of a specific product or application.

Further information on the DECUS Library, publications, and other DECUS activities is available from the DECUS offices listed below:

DECUS
Digital Equipment Corporation
146 Main Street
Maynard, Massachusetts 01754

DECUS EUROPE
Digital Equipment Corp. International
(Europe)
P.O. Box 340
1211 Geneva 26
Switzerland

Please cut along this line.

READER'S COMMENTS

NOTE: This form is for document comments only. Problems with software should be reported on a Software Problem Report (SPR) form (see the HOW TO OBTAIN SOFTWARE INFORMATION page).

Did you find errors in this manual? If so, specify by page.

_____

_____

_____

_____

_____

_____

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

_____

_____

_____

_____

_____

_____

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

_____

_____

_____

_____

_____

_____

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

If you do not require a written reply, please check here. ☐

---------- Fold Here ----------

---------- Do Not Tear - Fold Here and Staple ----------

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Software Communications
P. O. Box F
Maynard, Massachusetts 01754