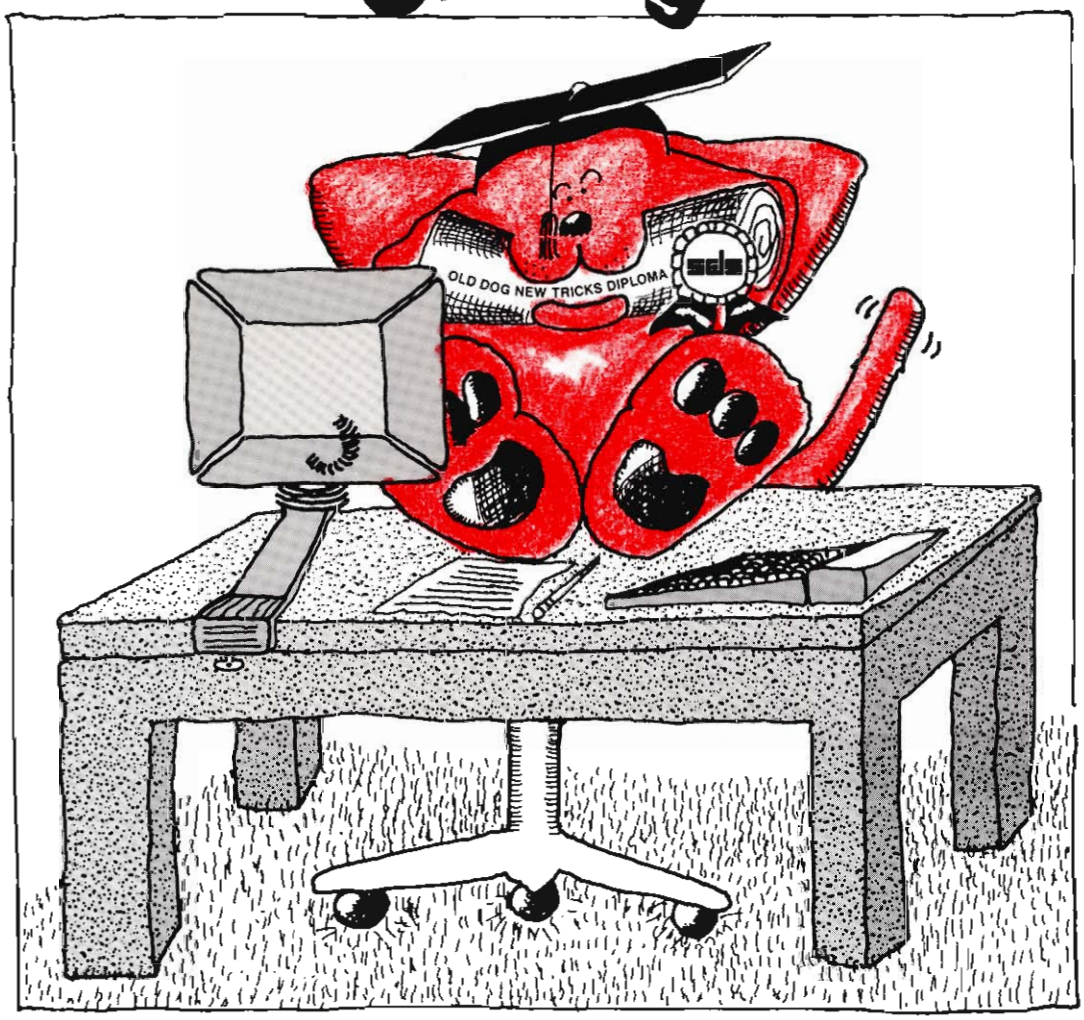
The 2200 Old Dog



Training Guide

A brief description of SDS-Extended BASIC-2 Release 4

sds southern data systems, inc.
5115 Holly Ridge Drive Raleigh, NC 27612 919-781-7603

SDS-Extended BASIC-2

Release 4.0

SOFTWARE DESCRIPTION

Multiuser Operating System
and
BASIC-2 Language Interpreter

developed by
Southern Data Systems, Inc.
for
Wang 2200 Series Processors

Fifth Edition (4.0.1) -- March, 1987
Fourth Edition (4.0.0) -- December, 1986
Third Edition (2.8.2) -- January, 1986
Second Edition (2.8.1) -- June, 1985
First Edition (2.8.0) -- February, 1985

DISCLAIMER OF WARRANTIES AND LIMITATION OF LIABILITIES

The staff of Southern Data Systems, Inc., has taken due care in preparing this manual; however, nothing contained herein modifies or alters in any way the standard terms and conditions of the Southern Data Systems, Inc. license agreement by which this software was acquired, nor increases in any way Southern Data System's liability to the customer. In no event shall Southern Data Systems, Inc. be liable for incidental or consequential damages in connection with or arising from the use of this manual, the associated equipment, any software programs contained herein, or any related materials.

PREFACE

This Software bulletin describes SDS-Extended BASIC-2 Release 4.0, its installation procedures and the software features of the Multi-user Operating System and BASIC-2 Language Interpreter for Wang 2200 processors.

Chapter 1 describes SDS-Extended BASIC-2 and the installation procedures for Release 4.0.

Chapter 2 describes the new features and the BASIC-2 language enhancements provided in Release 4.0:

- a new #HASH function
- a new #ID' function
- a new #LINE function
- a new #OPEN function
- a new $HELP statement
- a new $ID function
- a new $RELEASE statement
- a new =SELECT statement
- a new Multi-byte BIN function
- a new DATA LOAD BA enhancement
- a new DATA SAVE BA enhancement
- a new ERR$ command
- a new GOSUB "load-module" statement
- a new HEXPRINT+ statement
- a new HEXPRINT- statement
- a new IF -END statement
- a new 'selective' LIST DC command
- a new LIST DT output format
- a new LIST L statement
- a new LIST STACK statement
- a new LIST+V command
- a new LOAD "load-module" statement
- a new RENAME command
- a new RESAVE command
- a new SELECT STOP command
- a new TRACE V command
- a new Multi-byte VAL function
- Descriptive Error Messages
- Time and Date Stamp on program files
- UPPER/lower case commands
- Using variables for line-numbers
- Redirecting PRINT output to disk (SPOOL)
- The SDS RAM/Disk feature
- Up to 16MB Data Memory support

This documentation is intended to be used in conjunction with the following manuals:

- Wang BASIC-2 Language Reference Manual
- Wang BASIC-2 Disk Reference Manual
- Wang BASIC-2 Utilities Reference Manual
- Wang Release 2.5 Software Bulletin

CONTENTS

CHAPTER 1

SDS-Extended BASIC-2

What is SDS-Extended BASIC-2?

SDS-Extended BASIC-2 is a multi-user operating system and enhanced BASIC-2 language processor the purpose of which is to extend the usefulness, the effectiveness, the efficiency and the user friendliness of the Wang 2200 Computer System.

SDS-Extended BASIC-2 Release 4.0 adds significant new capabilities to the 2200 system. For example, you can now RENAME files, RESAVE programs, LIST the system STACK, and SPOOL print output, just to mention a few. Release 4.0 also adds the capability to load and execute machine language subroutines directly from a BASIC-2 program. The performance improvements possible via execution of machine language subroutines could mean that your 2200 system might run certain applications up to 40 times faster.

Release 4.0 is the first operating system which allows utilization of up to 64K of control memory. However, 2200 users with only 32K of control memory can still benefit from all the features of Release 4.0, including the 'Load Module' feature, since Release 4.0 will operate with a minimum of 32K of control memory.

What does SDS-Extended BASIC-2 run on?

SDS-Extended BASIC-2 operates on all Wang 2200 series SVP, MVP, LVP, MVPC, LVPC and MicroVP processors. One of these processors with a minimum of 32K of Control Memory and your registered copy of SDS-Extended BASIC-2 is all that is required.

How to get your copy of SDS-Extended BASIC-2

SDS-Extended BASIC-2 demonstration diskettes are distributed to users which allow limited-use of the software for evaluation purposes. Your demonstration copy may be converted to a full-use copy and registered as a licensed and personalized installation by telephone. A simple registration program which you run during this phone call is included on the demonstration diskette.

During this registration process, an authorization number will be provided and your licensed copy of SDS-Extended BASIC-2 will be created with your unique registration number and your name encoded in the machine code on your system. You then simply re-initialize your system and start enjoying the benefits of SDS-Extended BASIC-2.

Installing SDS-Extended BASIC-2

Two steps are required to utilize SDS-Extended BASIC-2:

1- Install the limited-use demonstration software on your system.

2- Convert your demonstration copy to a full-use, licensed copy, personalized for your system.

CAUTION

It is very important that you make a BACKUP COPY of your system files before running the INSTALL program.

To install the demonstration copy of SDS-Extended BASIC-2, you run the 'INSTALL' program included on the distribution diskette. The 'INSTALL' program moves the operating-system/language-interpreter module and an 'SDS-Preloader' module to your initialization disk.

If the installation process is interrupted for any reason... a power fluctuation, for example... you may not be able to use the partially installed version of the operating system to reinitialize your system. You must restore your original operating system files and begin the installation process again.

The contents of the demonstration disk are:

INSTALL	Installs limited-use Demonstration copy
@LICENSE	Creates full-use, licensed, personalized copy
@SDS	Operating System/Language Interpreter
@MVP	SDS-Preloader Module
HELP	HELP text file for $HELP verb
@GENPART	Modified partition generation program which supports 1MB and larger memory

The Results of Running the INSTALL Program

The INSTALL program renames your existing operating system files and installs the new SDS-Extended BASIC-2 files. The following new or renamed files are created on your initialization disk:

@MVPold	Renamed copy of your old @MVP file
@MVP	SDS-Preloader Module
@SDS	Operating System/Language Interpreter

The INSTALL program does not move the '@LICENSE', 'HELP' or '@GENPART' files. You may move these files to any disk in your system.

Installation Procedure

ACTION	COMMENT
1. Make a backup copy of your current operating system.	Store it in a safe place.
2. Type: LOAD RUN T/Daa, "INSTALL"	where "Daa" is the disk address where the SDS-Extended BASIC-2 demonstration software is located.
Press RETURN	SDS-Extended BASIC-2 installation screen is displayed.
Press any key	An input prompt is given at the bottom of the screen, "Enter the Device Address of the disk containing your operating system"
3. Type in the Device Address of your system disk. Press RETURN	A second input prompt is given, "Enter the Device Address of the disk containing SDS-Extended BASIC-2.
6. Type in the Device Address of the disk where the SDS-Extended BASIC-2 demonstration software is located. Press RETURN	A limited-use, demonstration copy of SDS Extended BASIC-2 is installed on your disk.

When the INSTALL program is complete the display will read,

"Installation has been successfully completed"

How to create and license your full-use, personalized copy

You (or your software consultant) may license SDS-Extended BASIC-2 for your system by running the '@LICENSE' program included on the demonstration copy disk and calling Southern Data Systems for a registration and authorization number. The '@LICENSE' program will create your personalized, full-use copy to run on your system.

Initializing your system under SDS-Extended BASIC-2

In order to utilize the features of SDS-Extended BASIC-2, you must re-initialize your system using the new operating system files installed by the 'INSTALL' program. SDS-Extended BASIC-2 is designed so that the start-up procedures are the same as previous releases of the operating system.

You should initialize ('boot') your system from Terminal-1 just as you usually do. The following procedure will initialize your system for operation under SDS-Extended BASIC-2:

ACTION	COMMENTS
1. Mount the disk which contains SDS-Extended BASIC-2	This disk may already be mounted if you specified a fixed disk in the INSTALL program.
2. Master Initialize the system from the disk which contains SDS-Extended BASIC-2	This allows the system startup menu program to display a selection of "DIAGNOSTICS" or "BASIC-2" on your screen.
3. Select "BASIC-2" Press RUN.	The SDS-Preloader displays the SDS-Extended BASIC-2 Registration Screen for about 10 seconds while it loads the operating system.

For licensed, full-use versions, the SDS-Extended BASIC-2 operating system loads and runs the partition generator program '@GENPART'. This will either automatically configure the system for you or the interactive system configuration screen will be displayed, depending on the options set up on your system. You should complete this operation as you usually do.

For limited-use demonstration copies, the SDS-Extended BASIC-2 operating system will also load and run the '@GENPART' program, however, it will limit configurations to a maximum memory of 61K in bank 1 and up to 4 terminals. If your system is set up for automatic configuration, you should create a partition configuration meeting this restriction before initializing under a demonstration version of SDS-Extended BASIC-2.

CHAPTER 2

FEATURES OF RELEASE 4.0

What new features and enhancements are included?

The following pages describe the new features and the BASIC-2 language enhancements in SDS-Extended BASIC-2 Release 4.0. The format and style of the function descriptions is presented in the familiar form used by the BASIC-2 Language Reference Manual.

The features and enhancements covered are:

- a new #HASH function
- a new #ID' function
- a new #LINE function
- a new #OPEN function
- a new $HELP statement
- a new $ID function
- a new $RELEASE statement
- a new =SELECT statement
- a new Multi-byte BIN function
- a new DATA LOAD BA enhancement
- a new DATA SAVE BA enhancement
- a new ERR$ command
- a new GOSUB "load-module" statement
- a new HEXPRINT+ statement
- a new HEXPRINT- statement
- a new IF -END statement
- a new 'selective' LIST DC command
- a new LIST DT output format
- a new LIST L statement
- a new LIST STACK statement
- a new LIST+V command
- a new LOAD "load-module" statement
- a new RENAME command
- a new RESAVE command
- a new SELECT STOP command
- a new TRACE V command
- a new Multi-byte VAL function
- Descriptive Error Messages
- Time and Date Stamp on program files
- UPPER/lower case commands
- Using variables for line-numbers
- Redirecting PRINT output to disk (SPOOL)
- The SDS RAM/Disk feature
- 1MB and larger memory support

#HASH function

General Form:

... #HASH (alpha-exp, [end-char], [type-hash], modulus) [...]

Where: alpha-exp = a literal-string or alpha-variable containing the expression to be evaluated.

end-char = An optional literal-string or alpha-variable the first byte of which indicates that the hash calculation should terminate at the first occurance of this value in the alpha-expression. If omitted, the hash calculation will continue for the entire length of the alpha-expression.

type-hash = an optional numeric value or numeric-expression indicating the hash algorithm to be used. A value of 0 invokes the original disk index hash algorithm, 1 invokes the new alternate disk index (see SCRATCH') algorithim. If omitted, 0 is assumed.

modulus = A numeric value or numeric-expression greater than zero and less than 65536 indicating the number of values into which the alpha-expression must be hashed. The maximum value is dependent on the hash type specified.

Purpose:

A built-in function that returns the hash value of an alpha-expression over a given modulus. Two hash algorithms are available, the same ones used by the operating system to locate entries for the disk index.

Examples:

```
100 X=#HASH ("SDSBASIC",,1,97)

200 Y=#HASH (V$,HEX(20),,255)

300 B=155:                      REM number of buckets
310 LINPUT"Enter Key",-K$:      REM key value
320 K= #HASH (K$,,,B)           REM calculate bucket
```

#ID' Function

General Form:

... #ID' [...]

Purpose:

The #ID' function returns the value of the SDS-Extended BASIC-2 Operating System registration number. This value is a number between 0 and 999999 and is the same number that appears in <> brackets in the READY message. This value is unique for each registered copy of SDS-Extended BASIC-2. This is not the same value as the CPU identification number returned by the #ID function. Both the #ID' function and the #ID function are useful in licensing software to specific users for operation on specific installations.

Examples:

```
PRINT #ID'
A=B+#ID'

10 IF #ID' <> 121617 THEN STOP

20 IF #ID' + #ID = 568821 THEN 30
 : PRINT "SYSTEM NOT LICENSED FOR USE IN THIS CONFIGURATION"
 : PRINT "PLEASE CALL SOUTHERN DATA SYSTEMS, INC."
 : STOP
30 PRINT "Your CPU number is ";#ID
 : PRINT "Your OS Regristration number is ";#ID'
```

#LINE function

General Form: ... #LINE [...]

Purpose: A built-in function that returns the line number of the current line in the program being executed. The value returned is from 0 to 9999 corresponding to the number of the current program line being executed.

Examples:
```
100 X=#LINE

PRINT #LINE
```

#OPEN function

```
                                [   f#                ]
General Form:      ... #OPEN [[/]taa                ] [...]
                                [   <alpha-variable>]
```

Where:

f# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15. If omitted, slot #0 is assumed.

taa = An explicit disk-address where t is the device-type and aa is the unit-address.

Purpose: A built-in function that returns the partition number of the partition currently hogging the specified device. A value of 0 is returned if the device is not currently hogged.

Examples:

```
100 X=#OPEN #7
110 PRINT "Device hogged by partition #";X

PRINT #OPEN /215

PRINT #OPEN <V$>
```

$HELP Function

General Form:

```
        [ file#,]
 $HELP[S][platter][<filename>][keyword][;][TO alpha-receiver]
        [ T/daa,]
```

Where:

keyword = an optional alpha-expression, the subject of the HELP request.

filename = an optional expression which defines the name of the help file to be searched.

; = allows continuous output of the help screens in the specified <filename> starting with the specified 'keyword' subject and continuing in keyword order.

alpha-receiver = An optional string-variable to receive the text message rather than having the message displayed on the screen.

Purpose:

The $HELP function provides a convenient means of retrieving information associated with a 'keyword' subject. The $HELP function searches the specified <filename> for the text associated with the 'keyword' and then transfers the text to either the screen (direct form) or to an alpha-variable receiver (indirect form).

If the <filename> parameter is omitted, the file defaults to the SDS-Extended BASIC-2 Language/2200 System "HELP" file.

If the 'keyword' subject parameter is omitted, a list of all keyword entries in the specified <filename> is displayed. The keywords are displayed in the same order that they exist in the keyword section of the file (the list is not sorted prior to display). If a 'keyword' subject is specified and continuous viewing mode (';') is indicated, the text messages, beginning with the selected keyword, are displayed in the same order as the remaining keywords in the keyword section.

Examples:

```
$HELP                              $HELP S "MAT MERGE"
$help #7, "$BREAK";                $Help T/D11,"#PART"
$helps <"Payroll"> "FICA";         $HELP,TO A1$()
```

```
20 $HELP <"GL HELP"> "ACCOUNT-NUMBER" TO D$()
30 DEFFN'0 (A$,B$): $HELP <B$> A$: RETURN
40 REM Display Payroll Screen:$HELP<"SCREENS">"PR Input Screen"
50 REM Display Operation Error Message:$HELP <"ERR-MSG"> E$(E)
60 REM Display Blank Order Entry Form:$HELP<"SCREENS">"OE-Input"
```

The HELP File Structure

Help message files consist of a 'keyword' section and a 'text' section. Both are written as BA formatted sectors. The keyword section may contain any number of entries which point to message starting points within the text section. Multiple keywords may point to the same message text.

The KEYWORD section consists of a one sector header record followed by as many keyword sectors as required by the help file. The header record sector and the keyword sectors are structured as follows:

Sector 1	xFILENAMExttttttttttttttttttttttt...tttttt
Sector 2	LkkkkkkkkkPPPLkkkPPPLkkkkkkPPP. . .LkkkkkkPPPd
Sector 3	LkkkkkkkPPPLkkkPPPLkkkkkPPP. . .LkkkkkPPPd
. . .	. . .
Sector n	LkkkkkPPPLkkkkkkkkkkkkkkkPPP . . .LkkkkkPPPf

where:

- x = Hex(20) control character
- FILENAME = name of help file
- tt..tt = optional string of 246 characters
- L = Length of keyword (binary value)
  - L < HEX(80) for primary keyword,
  - L > HEX(80) for secondary keywords which are skipped in continuous display mode unless it is the first keyword accessed
- kk..kk = ASCII encoded keyword
- PPP = 3 byte binary pointer to start of text for keyword (relative number of bytes from start of file)
- d = Hex(FD) code used to end current keyword sector
- f = Hex(FF) code used to mark end of keyword section

The TEXT section of a help-file may contain any data to be transfered to either an alpha-variable receiver or displayed on the CRT, including control codes. Each message consists of a two byte binary count which defines the length of the message followed by the message text. A single text message may span as many sectors as required to complete the message, however the binary character count for the message must not be split between two adjacent sectors.

A special multi-byte control sequence of HEX(020Bxxabcd) is defined to implement the PRINT BOX, TAB, and AT functions. The BOX, AT, and Erase control sequences are ignored if the selected list device is not a CRT. The following definitions explain the options available:

Sequence	Equivalent function	Definitions
020B01hhww	BOX(heigth,width)	hh = BIN(height), ww = BIN(width)
020B05hhww	BOX(-heigth,-width)	hh = BIN(height), ww = BIN(width)
020B02rrcc	AT(row,column)	rr = BIN(row), cc = BIN(column)
020B04cccc	Erase(count)	cccc = BIN(count,2)
020B03cccc	TAB(column)	cccc = BIN(column,2)

$ID Function

General Form:

alpha-variable = $ID

Where:

alpha-variable = any alphanumeric string variable receiver

Purpose:

The $ID function returns an alpha-numeric string equal to the name of the registered licensee of the resident copy of the SDS-Extended BASIC-2 Operating System. The $ID function can only appear on the right-hand side of alpha assignment statments. Programs can then use the string value whenever and however convenient.

Examples:

Assume the registered licensee is "SDS Distributing Co., of Raleigh, NC". Then the name line listed on the system Pre-loader screen displayed during system initialization would read:

"SDS Distributing Co., -- Raleigh, NC"

The following statements would give the results indicated.

```
DIMA$80
A$=$ID
PRINT A$
SDS Distributing Co., -- Raleigh, NC.

10  DIM A$80
20  A$=$ID
30  PRINT "This software is licensed to ";A$;" under
    operating system regristration number ";#ID';" for
    operation on CPU number ";#ID

70 REM Get Client Name
 : A$=$ID
 : N$=STR(A$,1,POS(A$="-")-1)
```

$RELEASE statement

General Form: `$RELEASE module-name`
`ALL`

Where:

module-name = A literal-string or alpha-variable containing the name of the load module to be removed from control memory.

ALL = A parameter specifying that all currently resident load modules are to be removed from control memory.

Purpose: Allows clearing a portion or all of the section of control memory allocated for user loading of machine language 'Load Modules'.

Examples:

```
$RELEASE "TURBO-SP"

10 $RELEASE "DATECONV"

80 $RELEASE ALL
```

=SELECT Function

General Form:

alpha-variable =SELECT parameter

Where:

alpha-variable = a string variable which will receive the value requested.

parameter = a keyword defined by the following table.

parameter	value returned	length of value returned	format of data returned
CI	Console Input device address	2	0taa
INPUT	INPUT device address	2	0taa
PLOT	PLOT device address	2	0taa
TAPE	TAPE device address	2	0taa
CO	Console Output device address	4	0taaww00
PRINT	PRINT device address	4	0tbbww00
LIST	LIST device address	4	0taaww00
#n	0<=n<=15 file-status parameters	8	ftaa ssss cccc eeee
ALL	Master Device Table for current partitions device selections	64	aaup ...

The symbols used in the data format column are defined below:

t = one hex digit specifying the device-type
aa = two hex digits specifying the physical device address
bb = two hex digits specifying the physical device address or Spool slot number
ww = two hex digits expressing the current maximum line width
f = file status (0=not open, 1/2=open on "F"/"R" drive)

ssss = four hex digits specifying the starting sector address
cccc = four hex digits specifying the current sector address
eeee = four hex digits specifying the ending sector address

u = one hex digit where the binary bits represent device status
bit 1 =1 - device is a disk device
bit 2 =1 - device is assigned to exclusive use of partition (p)
bit 3 =1 - device is currently in use by partition (p)
bit 4 =1 - device is currently hogged by partition (p)

p = one hex digit specifying the number of the partition using the device

Purpose:

Typically a program tracks device selection by setting up a variable or array and maintaining a copy of the current SELECT parameters for decisions on various program actions required by certain devices. An example is in a program producing printed output to either a printer or the the CRT screen. A variable would be set to tell the program to stop every 24 lines if the CRT were selected, or the skip to a new page every 60 or so lines if a printer were selected.

The =SELECT statement provides a convenient means of reading the values contained in the device table. This allows the program to know any SELECTED device at any time without having to maintain a seperate set of varaibles for this purpose.

Example:

```
10 DIM A$4
20 SELECT PRINT 215(132)
30 A$=SELECT PRINT
40 HEXPRINT A$

the printed output is

'02158400'
   ↑  ↑
   |  └── Width is Hex(84) = Dec(132)
   |
   └──── Device is 215
```

Multi-byte BIN function

General Form:

```
alpha-variable = [...] BIN(numeric-expression [,n]) [...]
```

Where:

n = 1 >= length parameter <= 6

Purpose:

The BIN function converts the integer value of a numeric-expression to an 'n' byte binary number. If 'n' is not specified, a one byte binary number is created.

Examples:

```
D = 52607
D$ = BIN(D,3)    (Sets D$ = HEX(00CD7F)

10 Z$(1) = D$ ADD BIN(Q+R*2)
20 STR(X$,4,5) = BIN(X,5)
```

DATA LOAD BA

General Form:

```
DATALOAD BA pd [file#,] (sector [,next]) [alpha-array          ]
               [/taa, ]                  [alpha-var; [<offset>]]
```

Where:

pd = Platter-designator (F, R or T).

file# = A device-table-slot reference <= 15. If omitted, slot #0 is assumed. The device table data is not updated by DATALOAD BA.

/taa = A disk-address where t is (3, D, or E) and aa is the unit-address.

sector = A numeric-expression or alpha-variable designating the address of the sector to be accessed.

next = A numeric-variable or an alpha-variable which receives the sector address of the next sector following the last sector read.

alpha-array = An alpha-array of at least 256 bytes in size. If larger, only the first 256 bytes are loaded with data from the specified sector.

alpha-var = An alpha-variable of any size. As many bytes of data as required will be loaded to fill the variable starting at the specified 'sector' plus the 'offset' in bytes.

offset = A numeric-expression which specifies the number of bytes to skip before starting to load data into the alpha-variable. The offset must be a positive value which will not cause the access to exceed the maximum sector on the disk.

Purpose:

1- Load one 256-byte disk sector into an alpha-array variable.
2- Load any amount of data from a disk into an alpha variable.

The single sector load version of the command is not changed. The following are examples of the variable length load version of the command.

Examples:

```
10 DATALOAD BA T /D50, (0) X$;
20 DATALOAD BA T (A,B) STR(D$,25,87);<25>
30 DIM Z$(32000)1
40 DATALOAD BA T /D13, (S$,R) Z$();<Q>
```

DATASAVE BA statement

General Form:

```
DATASAVE BA pd [$][f#,] (sector[,nextvar]) [alpha-array          ]
               [da,]                       [alpha-var; [<offset>]]
```

Where: pd = Platter-designator (F, R or T).

$ = A parameter specifying that a read-after-write verification is to be performed. An I99 error results if the read-after-write fails.

f# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15. If omitted, slot #0 is assumed.

da = An explicit disk-address of the form /taa where t is the device-type and aa is the unit-address.

sector = A numeric-expression or alpha-variable designating the address of the sector to be accessed. If an alpha-variable, the first two bytes are treated as a 16-bit, unsigned binary value.

nextvar = A numeric-variable or an alpha-variable which receives the sector address of the next sector following the last sector saved.

alpha-array = An alpha-array of any size. If the alpha-array is shorter than 256 bytes, the remainder of the sector is filled with undefined data. If larger, only the first 256 bytes are written.

alpha-var = An alpha-variable of any size. The contents of the alpha-variable will be saved to the disk starting at the specified 'sector' plus the 'offset' in bytes. If the save does not start or end on a sector boundary, the bytes before or after the alpha-variable are unchanged.

offset = A numeric-expression which specifies the number of bytes to skip from the beginning of the sector specified before starting to save data to the disk.

PURPOSE:

1- SAVE one 256-byte disk sector from an alpha-array variable.
2- SAVE any amount of data to a disk from an alpha variable.

EXAMPLES:

```
10 DATASAVE BA T (S) X$()
20 DATASAVE BA T /D11, (S$,S$) Y$()
30 DATASAVE BA T #X(4), (N+3*M) STR(Z$(),100)
40 DATASAVE BA T (A,B) STR(D$,25,87);<25>
50 DIM Z$(32000)1
 : DATASAVE BA T /D13, (S$,R) Z$();<Q>
```

ERR$ Command

General Form:

alpha-variable = ERR$ (error-code)

Where: error-code = the 2-digit numeric portion of the error code.

Purpose:

The ERR$ statement provides a convenient means of providing you with an English description of an error. The ERR$ statement places the description of the error code requested in the alpha-variable specified.

When used in conjunction with the ERR function, a program can describe the most recent error to the operator with an appropriate recovery instruction.

Example:

```
100 DIM E$80

200 DATASAVE DC #1, A$(),B$
  : ERROR GOSUB '100
210 . . .

700 DEFFN '100
  : E$=ERR$(ERR)
  : PRINT AT(24,0,80);E$
  : ON ERR-80 GOTO 780,781
  : RETURN

780 REM Error Handler Routines
```

If an error occurs following execution of the DATASAVE DC statement on line 200 a branch occurs to Subroutine DEFFN '100 at line 700 which displays the description of the error on line 24 and branches to an error handler routine starting at line 780.

GOSUB statement

General Form:

Form 1: GOSUB line-#

Form 2: GOSUB name-sub [([*]num-arg [,[*]num-arg] ...)]
argument argument

Where: line-# = A numeric constant, scaler-variable, array-element or numeric expression whose integer value represents the program line-number beginning the subroutine.

name-sub = A literal-string or alpha-variable which identifies the machine language subroutine.

* = An optional parameter which, when used with numeric scalar-variables or numeric array-variables, indicates that the address of the variable is to be passed to the subroutine.

num-arg = A numeric scaler-variable, or numeric array-variable.

argument = A numeric constant or numeric expression. An alpha literal, scaler-variable, array-element, or array designator.

Purpose:

Form 1 of GOSUB causes execution to be transferred to the specified line number. Upon encountering a RETURN statement, the system transfers execution to the statement immediately following the GOSUB. Form 1 subroutines may call other subroutines.

Form 2 of GOSUB causes execution to be transferred to the machine language subroutine identified as 'name-sub' in a currently active load module in control memory. If the cannot be located a P31 error is reported. Upon completion of the subroutine, execution transfers to the next statement following the GOSUB.

GOSUB can optionally transfer values and/or pointers of the argument list to the machine language subroutine. The type of argument (value or pointer, numeric or alpha numeric) must be the same as the type defined for the load module.

Examples: Form-1
```
10 GOSUB 4000
20 GOSUB V
30 GOSUB 4000+X
```

Form-2
```
40 GOSUB "TEST-SYS"
50 GOSUB "DATECONVERT" (3,"022685",*X1,D1$)
60 GOSUB A1$ (*C,C4$,STR(G$,3,17))
```

HEXPRINT- and HEXPRINT+

General Form:	Form 1:	HEXPRINT- (alpha-variable) (literal-string)
	Form 2:	HEXPRINT+ (alpha-variable) (literal-string)

Purpose:

The regular HEXPRINT statement is used to print the value of an alpha-variable or literal-string in hexadecimal notation. The format of the printed value is a continuous string of hexadecimal characters and frequently difficult to read.

The HEXPRINT- statement also prints the value of an alpha-variable or literal- string in hexadecimal notation, however, the format is changed to show pairs of hexadecimal digits seperated by spaces.

The HEXPRINT+ statement prints the value of an alpha-variable or literal-string in both hexadecimal and ASCII format. The printed output is arranged in typical dump format with up to 16 hexadecimal characters seperated by spaces followed by their equavalent ASCII characters for each line. Values which do not convert to printable ASCII characters are displayed as periods.

Examples:

Assume: DIM A$32
A$="SOUTHERN DATA SYSTEMS, INC."

then: HEXPRINT A$ prints:

534F5554845524E2044415441205350535445 4D532C20494E432E2020202020

HEXPRINT- A$ prints

53 4F 55 54 48 45 52 4E 20 44 41 54 41 20 53 59 53 54 45 4D 53 2C
20 49 4E 43 2E 20 20 20 20 20

HEXPRINT+ A$ prints

```
53 4F 55 54 48 45 52 4E 20 44 41 54 41 20 53 59   SOUTHERN DATA
53 54 45 4D 53 2C 20 49 4E 43 2E 20 20 20 20 20   SYSTEMS, INC.
```

IF -END THEN Statement

General Form:

```
IF -END THEN (line-number)[ELSE statement]
              (statement  )
```

Purpose:

The IF END THEN statement is used to test for the presence of an end-of-file record when reading records from a disk.

The IF -END THEN statement is used to test for the ABSENCE of an end-of-file record.

Examples:

Using IF END

```
100 DATALOAD DC A,B,C$
110 IF END THEN 130
120 PRINT A,B,C$
130 . . .
```

Using IF -END

```
100 DATALOAD DC A,B,C$
110 IF -END THEN PRINT A,B,C$
.
130 . . .
```

LIST DCT "mask" Command

General Form:

```
                                        [<D> ]
                                        [<P> ]   ["mask"          ]
                    LIST [S] DC platter [<S> ][-][alpha-variable]
                                        [<SD>]   [(sector)        ]
                                        [<SP>]
```

Where:

<D>	selects only data files for the list
<P>	selects only program files for the list
<S>	selects only scratched files for the list
<SD>	selects only scratched data files for the list
<SP>	selects only scratched program files for the list
'-'	a minus sign ('-') reverses or negates the selection logic
mask	a literal-string or an alpha-variable to define the files to be selected. The mask can contain exact match characters or by using a leading asterik "*", the search will look for any occurance of the mask-string in the file name. Don't-care character positions are defined by the "?" character.
sector	a starting sector (or ending sector if a minus is used) will cause the list to begin (or end) at this sector number.

Purpose:

The LIST DC command now allow you to search a disk for selected groups of files. Files may be grouped by name (mask), type (program, data), status (scratched, active) or location (before/after a specified sector).

Examples:		
	LIST SDCT/310, "GL"	lists all files with "GL" characters in positions 1 and 2
	LIST S DCT "*ASK"	lists all files which have the "ASK" character string in any position of their name
	LIST S DCT "??AB??"	lists all files which have "AB" as the third & forth characters.
	LIST S DCT <SD>(2300)	lists all scratched data files residing above sector 2300.
	LIST S DCT <D>-"*@"	lists all data files except ones with an "@" character anywhere in their name.

LIST DT output format

General Form:

LIST [S] [title] DT

Where:

title = any literal-string or alpha-variable (not displayed if List device is CRT)

Purpose:

The LIST DT command displays the current contents of the device table in both decimal and hexidecimal notation. The table is displayed in the following format:

Console Input Plot Tape Console Output Print Output List Output

```
CI-taa  IN-taa  PL-taa  TA-taa  CO-taa ll www  PR-taa ll www  LT-taa ll www
#  Dev   Start      Current      End    S
 0 taa ssss sssss cccc ccccc eeee eeeee y
 1 taa ssss sssss cccc ccccc eeee eeeee y
 2 taa ssss sssss cccc ccccc eeee eeeee y
 3 taa ssss sssss cccc ccccc eeee eeeee y
 4 taa ssss sssss cccc ccccc eeee eeeee y
 5 taa ssss sssss cccc ccccc eeee eeeee y
 6 taa ssss sssss cccc ccccc eeee eeeee y
 7 taa ssss sssss cccc ccccc eeee eeeee y
 8 taa ssss sssss cccc ccccc eeee eeeee y
 9 taa ssss sssss cccc ccccc eeee eeeee y
10 taa ssss sssss cccc ccccc eeee eeeee y
11 taa ssss sssss cccc ccccc eeee eeeee y
12 taa ssss sssss cccc ccccc eeee eeeee y
13 taa ssss sssss cccc ccccc eeee eeeee y
14 taa ssss sssss cccc ccccc eeee eeeee y
15 taa ssss sssss cccc ccccc eeee eeeee y
taa-ppx taa-ppx taa-ppx taa-ppx taa-ppx taa-ppx taa-ppx taa-ppx taa-ppx taa-ppx
taa-ppx taa-ppx taa-ppx taa-ppx taa-ppx taa-ppx taa-ppx taa-ppx taa-ppx taa-ppx
taa-ppx taa-ppx taa-ppx taa-ppx taa-ppx taa-ppx taa-ppx taa-ppx taa-ppx taa-ppx
taa-ppx taa-ppx
```

Where:

taa = (t) device-type, (aa) unit device address
ll www = line width (ll=hexidecimal, www=decimal)
ssss sssss = starting sector address (hex and decimal)
cccc ccccc = current sector address (hex and decimal)
eeee eeeee = ending sector address (hex and decimal)
y = file open status (0=not open, 1=open on F, 2=open on R)
pp = number of partition using device
x = X if device is opened for exclusive use of partition pp
0 if device is opened by partition pp

LIST L statement

General Form: LIST [S] [title] L

Where: S = Specifies that the listing is to be output a set number of lines at a time as determined by SELECT LINE. Press RETURN to continue the listing.

title = An alpha-variable or literal-string. If included, title causes the system to send a top-of-form prior to printing the highlighted title and program text.

Purpose:

To produce a listing of all 'load-modules' and the machine language subroutines which are currently available in control memory. The subroutines listed may be called using the GOSUB statement.

Individual 'load-modules' may be removed from control memory by the $RELEASE 'load-module' statement. All currently active 'load-modules' are removed by the $RELEASE ALL statement.

LIST STACK statement

General Form: LIST [S] [title] STACK

Where: S = Specifies that the listing is to be output a set number of lines at a time as determined by SELECT LINE. Press RETURN to continue the listing.

title = An alpha-variable or literal-string. If included, title causes the system to send a top-of-form prior to printing the highlighted title and program text.

Purpose:

To produce a listing of all currently active FOR/NEXT loops and GOSUBs in the interpreter program stack. The actual program lines which created the stack entries are displayed. The most recent stack entry is displayed first.

In the case of FOR/NEXT loops, the current value of the index variable, the end value and the STEP value are displayed immediately following the FOR/NEXT statement.

LIST +V command

General Form:

```
LIST [S] [title] +V [variable-name] [,[variable-name]
```

Where:

variable-name = any valid BASIC-2 variable name

title = alpha-variable or literal-string

Purpose:

The LIST +V command generates a listing of the currently defined variables, an indicator of variable type (N=non-common, C=common), their current dimensions and their current value(s). The list may include one varaible, a range of variables, or all variables. For array variables, as many element values as will fit on the screen width are displayed.

Examples:

Assume the following program is currently loaded:

```
 5 COM Z$(14,2)10
10 DIM A(2,3),A$(4),B$28,C(4)
20 A(1,1)=9
30 B$="Southern Data Systems, Inc."
40 X=A(1,1)+Q
50 A$(1)=STR(B$,1,13)
60 C(3)=X*3
```

LIST V Produces cross-reference list

```
A(      - 0010 0020 0040
A$(     - 0010 0050
B$      - 0010 0030 0050
C(      - 0010 0060
Q       - 0040
X       - 0040 0060
Z$(     - 0005
```

LIST +V Produces dimension/value list

```
A(   - N (2,3)      9 0 0 0 0 0
A$(  - N (4)16      Southern Data
B$   - N 28         Southern Data Systems, Inc.
C(   - N (4)        0 0 27 0
Q    - N            0
X    - N            9
Z$(  - C (14,2)10
```

LOAD "load-module" statement

```
General Form:     LOAD [DC] pd [file#,  ] file-name
                               [address,]

Where:       pd = Platter-designator (F, R or T)

          file# = A device-table-slot reference of the form #nn
                  where nn is a numeric-expression such that 0 <=
                  nn <= 15.  If omitted, slot #0 is assumed.

       address = An explicit disk-address of the form /taa where
                  t is the device-type and aa is the
                  unit-address.

     file-name = A catalogued, machine language, control memory
                  load-module name.
```

Purpose:

To load a machine language 'load-module' into control memory from disk using disk catalogue mode. This LOAD statement is identical in format to normal BASIC-2 program loads except the operating system detects that the load is a 'load-module' by the file type byte in the program header sector and directs the load into control memory.

The 'load-module' subroutine entry names are located and made available to the user via Form-2 of the GOSUB statement. Each 'load-module' may support up to 15 named machine language subroutines..

Examples:

```
      LOAD T "subsset"
      LOAD DC T /D12, "DATECONV"
10 LOAD T "ISAM"
20 LOAD T#1, "ideas++"
30 LOAD DC T/D21, "kfamsubs"
40 LOAD T "turbosub"
```

RENAME Command

General Form:

```
RENAME [DC] pd [f#,] (oldname) newname
               [da,]
```

Where:

DC = An optional parameter indicating disk catalogue mode. If omitted, DC mode is assumed.

pd = Platter-designator (F, R or T).

f# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15. If omitted, slot #0 is assumed.

da = An explicit disk-address of the form /taa where t is the device-type and aa is the unit-address.

oldname = A literal-string or alpha-variable containing the name of the existing file.

newname = A literal-string or alpha-variable containing the new name to be assigned to the existing file.

Purpose: The RENAME command provides a convenient means of changing an existing file name in a disk catalogue.

If the "oldname" does not exist on the specified disk a D82 error is reported. If the "newname" already exists on the specified disk a D83 error is reported.

Examples:

```
RENAME T ("PROGRAM1")"PROGRAM2"

10 RENAME T/D13,("GL-Acct1")"GL-Reorg"
20 RENAME T/D13,(A$)B$
```

RESAVE Command

General Form:

```
RESAVE [DC][<S >] pd [$][f#,] [!] filename [start][,end][/com]
           [<SR>]       [da,] [P]
```

Where:

DC = An optional parameter indicating disk catalogue mode. If omitted, DC mode is assumed.

<S> = A parameter indicating that all unnecessary spaces will be removed.

<SR> = A parameter indicating that all unnecessary spaces and REM statements will be removed.

pd = Platter-designator (F, R or T).

f# = A device-table-slot reference of the form #nn where nn is a numeric-expression such that 0 <= nn <= 15. If omitted, slot #0 is assumed.

da = An explicit disk-address of the form /taa where t is the device-type and aa is the unit-address.

$ = A parameter specifying that a read-after-write verification is to be performed.

filename = A literal-string or alpha-variable containing the name of the file.

! = A parameter indicating that the program text is to be scrambled.

P = A parameter indicating that the protect bit is to be set in the program file.

start = The starting line-number of text to be saved. If omitted, the lowest line-number is assumed.

end = The last text line-number to be saved. If omitted, the highest line-number is assumed.

com = An alpha-variable or literal-string containing up to 234 characters of information to be stored in the program header sector, immediately following the time and date stamp.

Purpose: The RESAVE command provides a convenient means of updating a currently existing program on disk with the program in memory. The RESAVE command performs the same functions the SCRATCH and SAVE commands.

If the "filename" does not exist on the specified disk a D82 error is reported. If the program in memory is larger than the "filename" file on the disk a D81 error is reported.

Examples:

```
10 RESAVE T "PROGRAM"
20 RESAVE DCF$ Q$
30 RESAVE <SR> T/D12, "FILENAME"/"New version"
40 RESAVE <S> T$#4, ! Y$ 1000,9000
```

SELECT STOP statement

General Form:

```
SELECT [... ,] STOP [[start-line],[end-line]] [...]
```

Where:

start-line = The lowest line number of the range of lines where HALT/STEP mode is to be automatically invoked. If omitted, the lowest line-number is assumed.

end-line = The highest line number of the range of lines where HALT/STEP mode is to be automatically invoked. If omitted, the highest line-number is assumed.

Purpose:

A new SELECT parameter to instruct the operating system to automatically place the partition in HALT/STEP mode when program execution is transferred to a line number within a specified range of lines. Upon program transfer to a line within the range, a 'TRANSFER TO xxxx' message is displayed and program execution is stopped.

Examples:

SELECT STOP (4000,9000)	Stop within 4000-9000 range
SELECT STOP (,2000)	Stop below 2000
SELECT STOP (8000,)	Stop above 8000
SELECT STOP	Clear stop function

TRACE Variable option

General Form:

TRACE [-T] [V variable-name]

Where:

variable-name = any currently defined variable

Purpose:

The TRACE V statement produces a trace of all operations which change the value of the specified variable.

In normal TRACE mode operation all program branches (FOR/NEXT, GOTO, GOSUB, RETURN) are identified as TRANSFER TO ### entries. The "-T" option inhibits the output of these transfer entries.

Trace output can be directed to either the CRT screen or to a printer by selecting the desired LIST device for output using the SELECT statement.

Examples:

TRACE V Q2	Traces variable Q2 only
TRACE V Z4$(	Traces array variable Z4$(only
TRACE -T	Traces all variables but suppresses the "TRANSFER TO XXX" entries in trace output
TRACE -T V X7	Traces variable X7 and suppresses the "TRANSFER TO XXX" entries in the trace output
TRACE OFF	Turns TRACE off and removes the V and -T options

Multi-byte VAL function

General Form:

```
VAL ([alpha-variable] [,n])
     [literal-string]
```

Where:

n = 1 >= length parameter <= 6

Purpose:

The VAL function converts the binary value of the first 'n' bytes of an alpha-string to a numeric value. If 'n' is not specified only the first byte is converted.

Examples:

```
PRINT VAL(A$,5)

X=VAL(D$,6)

D=VAL("010185",3)

10 IF VAL(X$,3) >262144 THEN 100
20 Y=VAL(STR(A$,14),4)
```

Descriptive Error Messages

Displays descriptive error messages on the screen

Purpose:

To provide on-screen display of error messages.

Example:

```
10 STR(A$(2,17,20)=STR(B$(J),2,20)
                  ^
                ERROR S11
Missing Right Parenthesis
```

Error Number	Description
1	Memory Overflow (Text <--> Variable Table)
2	Memory Overflow (Text <--> Value Stack
3	Memory Overflow (LISTDC, MOVE or COPY)
4	Stack Overflow (Operator Stack)
5	Program Line too long
6	Program Protected
7	Illegal Immediate mode statement
8	Statement not legal here
9	Program not Resolved
10	Missing Left Parenthesis
11	Missing Right Parenthesis
12	Missing Equal Sign
13	Missing Comma or mis-spelled atom
14	Missing Asterisk
15	Missing ">" character
16	Missing letter
17	Missing Hex Digit
18	Missing Relational Operator
19	Missing Required Word
20	Expected End of Statement
21	Missing Line Number
22	Illegal PLOT arguement
23	Invalid Literal String
24	Illegal Expression or Missing Variable
25	Missing Numeric-Scalar variable
26	Missing Array-Variable
27	Missing Numeric Array
28	Missing Alpha-Array
29	Missing Alpha-Variable
30	Not Currently Defined
31	Not Currently Defined
32	Start > End
33	Line-Number Conflict
34	Illegal Value
35	No Program in Memory
36	Undefined Line-Number or CONTINUE illegal
37	Undefined Marked Subroutine
38	Undefined FN Function
39	FNs Nested too Deep
40	No corresponding FOR for NEXT statement
41	RETURN with GOSUB
42	Illegal Image

Error Number	Description
43	Illegal Matrix Operand
44	Matrix not square
45	Operand dimesions not compatable
46	Illegal Microcommand
47	Missing Buffer Variable
48	Illegal Device Specification (Not in table)
49	Interrupt table full
50	Illegal Array Dimensions or Variable Length
51	Variable or Value too short
52	Variable or Value too long
53	Noncommon Variables Already defined
54	Common Variable required
55	Undefined Variable (Program not Resolved)
56	Illegal Subscript
57	Illegal STR Arguements
58	Illegal Field-Delimiter Specification
59	Illegal Redimension
60	Underflow
61	Overflow
62	Division by Zero
63	Zero divided by Zero or Zero Zero
64	Zero raised to a negative power
65	Negative number raised to Noninteger Power
66	Square root of a Negative Value
67	LOG of Zero
68	LOG of a Negative value
69	Arguement too large
70	Insufficient Data
71	Value exceeds Format
72	Singular Matrix
73	Illegal INPUT data
74	Wrong Variable type
75	Illegal number
76	Buffer exceeded
77	Invalid Partition Reference
78	Not currently defined
79	Not currently defined
80	File not OPEN
81	File is Full
82	Requested File is not in Selected Disk Catalog
83	File already exists in Catalog
84	File is not Scratched
85	Index is Full
86	Catalog END error
87	No End of File
88	Wrong Record Type
89	Sector Address beyond End of File
90	Incorrect or no response during Selection seq
91	Disk may not be mounted
92	Timeout Error
93	Disk Header Format problem
94	Format key engaged
95	Device fault
96	Disk Data error
97	Longitudinal Redundancy Check error
98	Illegal Sector Address or Platter not Mounted
99	Read after Write Error

TIME & DATE STAMP on Program Files

General Form:

SAVE [---standard parameters ---] [/text-string]

Where:

text-string = an optional literal-string or variable of up to 234 characters in length which is written in the program header sector on the disk.

Purpose:

If the system has an MXE terminal controller or is an SVP with an Option-W, the save statement will automatically write the DATE and TIME in the program header. If the DATE and TIME are not available in the system, zeros are written in the record.

Additionally, an optional text-string of up to 234 characters can be written in the program header record each time a program file is saved. This will allow the programmer to track program revisions and descriptions of programs as required.

The program header can be read using a DATA LOAD BA statement. The format of the program header is as follows:

```
Record   xNNNNNNNNxddmmyyhhmmsstttttttttttttttt . . . ttttt
                ↑       ↑ ↑     ↑     ↑                    ↑
Byte            2       9 11    17    23                   256
```

N = Program name
x = control character
ddmmyy = date (day/month/year)
hhmmss = time (hour/minute/second)
ttt . . . ttt = text string of 234 characters

Examples:

```
SAVE DC T/310, "PROGRAM1"/"Revision 1.7 JRE add range check"


A$ = "Revision 1.8 by Jim Smith,  Modified customer history
data fields per Change Order number 123-J17"
SAVE T "PROGRAM2"/A$


LIMITS T "PROGRAM1",A,B,C,D
DATA LOAD BA T (A), A$()
PRINT STR(A$(),22,234)      Prints comment saved with program
```

UPPER/lower case commands

General Form:

Commands and statements can be entered using either UPPER or lower case characters.

Purpose:

This feature allows commands and statements to be entered using either UPPER or lower case characters. All lower case commands are automatically converted to UPPER case by the operating system. Literal strings enclosed in quotes (" ... ") are not modified.

Examples:

lower case	Converted to
list sd	LIST SD
Print a$	PRINT A$
$release part	$RELEASE PART
convert X$ to X	CONVERT X$ TO X
select list 215	SELECT LIST 215
100 rem Start Search	100 REM Start Search
110 str(a$,1,4)="Sort"	110 STR(A$,1,4)="Sort"

Using variables for line-numbers

General Form:

```
for example:  GOSUB variable
              GOTO variable
              IF ... THEN variable
              KEYIN X$ (variable1,variable2)
              ON ... GOTO variable ...
              ON ... GOSUB variable ...
              PRINTUSING variable. ...
              RESTORE LINE variable
```

Purpose:

The destination (target) line number of many statements may be expressed as a numeric-variable or a numeric-variable expression. In the case of program flow control statements such as GOTO, GOSUB, IF-THEN and KEYIN, this allows a programs execution sequence to be modified by calculated values. In the case of line reference statements such as PRINTUSING and RESTORE LINE, this allows the referenced lines to be specified by calculated values.

Examples:

```
10 X=MOD(LEN(A$),4)
20 GOTO X

100 GOSUB Y+Z-2*Q+3000

200 RESTORE 300+LEN(Q$)*3

300 KEYIN K$ (X,Q)

400 PRINTUSING I,M$,D$,Y$
```

Redirecting PRINT output to disk (SPOOL)

General Form:

1- SELECT [... ,] SPOOL ['] #n, buffer-variable [...]

2- SELECT [... ,] PRINT SPOOL #n [...]

Where: ' = An optional parameter indicating that print data is to be compressed.

#n = A device table slot where 0 <= n <=15. If omitted, slot#0 is assumed.

buffer-variable = An alpha-array-variable >= 260 bytes to be used as a buffer for spooled print output.

Purpose:

To allow PRINT output to be placed in a disk file. The print output data (disk file) may be manipulated if required and printed later. It also facilitates development of print spooler functions and translater capabilities to support many different printers.

Operation Description:

Select statement 1 identifies a print output data buffer and assigns it to the previously opened device table slot #n. The spool mode may be specified as compressed or non-compressed by the ['] parameter. If compressed mode is specified, four or more consecutive occurences of the same character are compressed into three bytes in the form HEX(FBccxx) where 'FB' is the compression flag, 'cc' is the binary count and 'xx' is the HEX() value of the character. An exception is made for the occurence of the character HEX(FB) which is always expanded and saved as HEX(FBFB). Multiple occurences of HEX(FB) are never compressed.

Select statement 2 redirects print output data to the buffer associated with device table slot #n. The operating system will place print output data in the selected spool buffer until the buffer is filled at which time the contents of the buffer will be written to the disk in multiples of 255 bytes until the buffer contains less than 255 bytes. Disk interference can be controlled by the selection of buffer length.

Sectors will be written in BA mode with the device table parameters for slot #n being updated after each sector is written. Sectors contain one byte of data count and up to 255 bytes of print data. Buffer bytes 1-4 are used for system control. Bytes 1-2 are current data count. Bytes 3-4 are length of buffer. If compressed mode is selected, the value of HEX(80) is ORed into buffer byte 3.

In systems where document control information is saved in a header record, the initial contents of the buffer array may be preloaded and the current data count in bytes 1-2 may be adjusted prior to the spooling operation. Such document control information can then be used by a spool utility to manage the print documents stored on disk.

DSKIP and DBACKSPACE may be used to reposition the current file pointer to any position within the file. Attempts to write beyond the end of the file will result in a Dxx error and print output to the disk will be discontinued until the file is repositioned or a new file is opened.

A DATASAVE DC #n, END statement will cause any remaining buffer data and an END record to be written to the disk. A CLOSE operation on the file is optional.

Examples:

```
10 DIM B$(516)1
20 SELECT PRINT 215,#7 D13
30 DATALOAD DC OPEN T#7,"SPOOL-01"
40 SELECT SPOOL #7,B$()
50 SELECT PRINT SPOOL #7
 ... print statements
```

Any print statements following line 50 will be written to the buffer B$() and then to the "SPOOL-01" data file.

```
80 DATASAVE DC #7, END
```

Any remaining print data in the buffer, B$(), are saved to the disk, and an END record is written to the file associated with slot #7.

SDS RAM/Disk

General Form:

in any disk reference ... T/ED0, [...]

Purpose:

The RAM/Disk feature, available with 1MB or larger memory boards from Southern Data Systems, provides a separately addressable logical disk device that is actually resident in the unused portion of the system's data memory. The RAM/Disk will respond to all of the standard BASIC-2 disk commands similar to any other disk. Access times for data saved in RAM/Disk can be many times faster than for data stored on conventional disk drives.

The portion of memory not assigned to partitions during system initialization is automatically configured as RAM/Disk. The RAM/Disk responds to device code 'ED0' which can be SELECTed just as any other disk device. The presence of RAM/Disk and its size (in sectors) are available to the program through the 'SPACE DISK' function (similar to the 'SPACE' and 'SPACEK' functions).

Examples:

```
10 IF SPACE DISK >0 THEN
   SCRATCH DISK T/ED0,LS=5,END=SPACE DISK-1
20 DATA LOAD BA T/ED0,(A,X)A$()
30 MOVE T/D10, "PROGRAM" TO T/ED0,
40 COPY T/320,(0,1000) TO T/ED0,(0)
50 DATA SAVE DC T/ED0,A$(),B$()
60 $FORMAT DISK T/ED0,
70 SAVE T/ED0,(10)"NEWCODE"
80 SELECT #14ED0
```

Up to 16MB Data Memory Support

Release 4.0 includes modifications which allow 1MB or larger memory boards from Southern Data Systems to be utilized in 2200 systems. The operating system now provides the capability to address up to 16MB of Data Memory. With current memory technology, up to 4MB of memory is practical, however, as memory technology advances, larger memory configurations can be utilized without further changes to the OS.

The memory above 1MB provides a separately addressable logical disk device, called RAM/Disk, which responds to all of the standard BASIC-2 disk commands similar to any other disk. Access times for data saved in RAM/Disk can be many times faster than for data stored on conventional disk drives.

APPENDIX

The EZ-HELP Utility

A separate 'EZ-HELP' utility is available to assist users in creating 'help' message files. This utility consists programs which provide the functions of text entry, keyword entry and help-file creation.

Text messages can be designed on screen using word processing type operations, then keywords and/or key-phrases within the text can be identified and extracted to form the keyword section for the file. The sequence of keywords, which controls the order of presentation of the text messages when displayed in continuous mode, can be rearranged as desired before the file-builder creates the help-file.

Operating instructions for the EZ-HELP utility are provided by the $HELP function itself, through a special help-file, 'EZHELPIN". The 'EZ-HELP' utility can be ordered for operation on any system running SDS-Extended BASIC-2, Release 2.8.

Appendix B

The SDS-Extended BASIC-2 Language & 2200 System "HELP" file

The 'HELP' file supplied with SDS-Extended BASIC-2, Release 2.8 covers approximately 276 subjects associated with the 2200 language and system operation. Each screen (or page) consists of up to six sections. Sections are abbreviated or omitted where not appropriate.

1) Title and Reference
2) Purpose
3) Syntax
4) Comments and general usage rules
5) Examples
6) Other references

TITLE and REFERENCE The Title, the primary key used to access the page, is centered on the top line. The main Reference (a manual page number or description where further information on the keyword can be obtained) is right-justified within parenthese on the same line and take the following forms:

(number)	Page number, 1979 Wang BASIC-2 Language Reference Manual
(Disk number)	Page number, 1981 Wang BASIC-2 Disk Reference Manual
(Wang SB 2.x)	Wang BASIC-2 Software Bulletin Release 2.x
(SDS SB 2.x)	SDS-Extended BASIC-2 Software Bulletin

PURPOSE This is a brief description of the main purpose(s) of the keyword.

SYNTAX General syntax is specified showing correct command structure(s). Alternate and optional parameters and data are indicated. The following rules apply:

Upper-case	Must be used exactly as shown
Lower-case	Designate programmer-supplied information
"-" (hyphen)	Considered to be part of the programmer supplied information when imbedded in lower-case letters.
[ ] (brackets)	Indicate that the enclosed syntax is optional.
... (ellipsis)	The ellipsis has a dual meaning. When following an item enclosed in brackets, it indicates that the item may occur more than once. Otherwise, it indicates that other program text must (...) or may ... preceed or follow the specified syntax.

Syntax segments that are stacked vertically indicate that a choice is required. If the stacked segments are enclosed in brackets, the programmer may choose to eliminate the options entirely.

Example 1:

```
GOSUB line-number
```

The "GOSUB" is required exactly as shown; a "line-number" must be supplied by the programmer.

Example 2:

```
GOSUB' integer (argument, argument ...)
```

The "GOSUB" is required; the programmer must supply an "integer"; arguments are optional, but if included, they must be enclosed in parentheses; if arguments are used, at least one argument is required and additional arguments may be added, separated by commas.

Example 3:

```
... HEXOF (alpha-variable) ...
          (Literal-string)
```

In this case, program text must preceed the HEXOF function (the PRINT verb plus, optionally, other PRINT functions and separators); either an alpha-variable or literal-string is required, enclosed in paratheses; and additional text may optionally be added on.

Example 4:

```
LOAD RUN pd file#,    file-name
                      address,
```

Only LOAD RUN is required as all other parameters are optional. However, file# and address are mutually exclusive, and if used, must include the trailing comma.

COMMENTS — The section includes an explanation of all non-standard abbreviations used in the syntax, plus variable dimensioning requirements, parameter tables, restrictions, and information on the statement.

EXAMPLES — Most pages include one or more examples.

OTHER REFERENCES — Where appropriate, mention is made of other $HELP titles or manuals that contain more information, or which use the keyword in a different manner.

sds

SDS-Extended BASIC-2 Order Form and Software License Agreement

SDS-Extended BASIC-2 Software is distributed as fully functional Demo-software that contains a mechanism to limit the use of the software to demonstration and evaluation purposes. The Demo-software may be converted to Licensed-software on your system by telephone. To create and register a Licensed copy for your system you must complete this order form, read the license agreement on the reverse, sign below and return this form with full payment to SDS prior to receiving your authorization code.

Qty	Software Description	License Fee	Amount
	SDS-Extended BASIC-2 Release 4.0 (First CPU)	$249.00	
	Upgrade Release 2.8 to Release 4.0 (First CPU)	99.00	
	Upgrade Release 2.7 to Release 4.0 (First CPU)	149.00	
	Additional Authorized CPU(s)	99.00	
	Software update and support service for one year	49.00	

Qty	Media Description	Media Charge	Amount
	DSDD 5.25" Diskette (2275 drives)	no charge	
	SSSD 8" Diskette (2270A drives)	no charge	
	DSDD 8" Diskette (SVP/LVP drives)	no charge	
	Phoenix Pack (2280 drives)	175.00	
	31MB DATAPACK (SDS 2290-120 drives)	225.00	

Make check payable to Southern Data Systems, Inc. | Amount Due $

USER INFORMATION

Name

Address

Address

City State Zip

Contact Title

Phone ( ) PO No.

DEALER INFORMATION

Name

Phone ( ) Reg No.

NOTICE

Authorization codes required to activate full-use copies of SDS-Extended BASIC-2 will NOT be provided until this order form and license agreement is signed and returned with full payment to Southern Data Systems, Inc. 5115 Holly Ridge Drive Raleigh, North Carolina 27612

REGISTRATION DATA

For SDS use only

Date Who

Reg No. - - - - - - - #CPUs

Auth. No. - - - - - - - - - - - - - -

Acknowledgement and Signatures

USER acknowledges to have read this Software License Agreement, understands it, and agrees to be bound by its terms. USER further agrees that this agreement is the complete and exclusive statement of the agreement between the USER and Southern Data Systems, Inc. This Order and Software License Agreement is subject to acceptance by Southern Data Systems, Inc. which will be evidenced by return of a signed copy of this Order and Software License Agreement to USER, SDS being under no obligation to accept same.

USER

Printed Name _____

Signature _____

Title _____ Date _____

Accepted by SDS

Printed Name _____

Signature _____

Title _____ Date _____

SDS-Extended BASIC-2 Software License Agreement

1. This agreement is between Southern Data Systems, Inc., hereinafter referred to as 'SDS', and the USER identified on the reverse side and is entered into on the date herein written.

2. This agreement is for the purpose of granting use of certain proprietary computer programs and related materials which includes the SDS-Extended BASIC-2 software, the installation and authorization routines, other related software utilities and the associated documentation. Special versions of this software, hereinafter referred to as 'DEMO SOFTWARE', are intended for demonstration and evaluation purposes only and contain mechanisms to limit the use of the software. Customized versions of this software which do not contain any use limitations are hereinafter referred to as 'LICENSED SOFTWARE'. The term 'SOFTWARE' used herein shall mean both the DEMO SOFTWARE and the LICENSED SOFTWARE. A mechanism is provided to convert the DEMO SOFTWARE to LICENSED SOFTWARE at the USER site which requires an authorization code from SDS to de-activate the use limitations of the demonstration version. SDS provides the authorization code required to create the LICENSED SOFTWARE on the condition that the USER agrees to this license.

3. SDS agrees to grant and the USER agrees to accept a non-exclusive and non-transferable license to use the LICENSED SOFTWARE, including any subsequent updates, only on the designated computer system(s) identified at the time the LICENSED SOFTWARE is installed on the USER's computer system.

4. The USER agrees that no copies of the LICENSED SOFTWARE shall be made, except copies made for backup purposes. The USER agrees to place SDS proprietary notices on all copies of the LICENSED SOFTWARE. The USER may make additional copies of the DEMO SOFTWARE for distribution to third parties for demonstration and evaluation purposes only, provided that copies of this order form and license agreement are also provided to such third parties.

5. The SOFTWARE contains proprietary, confidential and trade secret information which is the property of SDS and SDS retains ownership rights to the SOFTWARE. The USER agrees not to modify any portion of the software. The USER agrees to notify SDS immediately of any unauthorized use or possession of the SOFTWARE.

6. In recognition of SDS proprietary rights and copyright protection of the SOFTWARE, the USER warrants and agrees that it will not inspect, disassemble, decompile, translate any portion of the licensed software into any other programming language, attempt to determine the internal methods of the SOFTWAREor attempt to create source programs for the SOFTWARE by reverse engineering of the SOFTWARE. The USER agrees that while this license is in effect, it will not directly or indirectly lease, license, sell, offer or negotiate to lease, license or sell, or contract for any software similar to that supplied under this license.

7. The USER agrees to pay to SDS, as full consideration for the LICENSED SOFTWARE, the license fees in effect at the time the LICENSED SOFTWARE is created plus any media and shipping charges and all applicable federal, state, or local taxes. If the support and update option is selected, USER agrees to pay, in addition to the license fee, the support fee in effect at the time and SDS agrees to provide any and all updates of the LICENSED SOFTWARE to USER during the support period.

8. The USER agrees that this Software License Agreement and the license granted hereunder may not be assigned or otherwise transferred by USER to any third party. In no event shall the SOFTWARE be subject to any levy, execution, attachment, garnishment, or seizure of any kind by any creditor, receiver, trustee in bankruptcy, or any other person, party, executor, successor or assign.

9. SDS warrants, for a period of six months after the date of the USER's creation of the LICENSED SOFTWARE for operation on the USER's system(s), that the LICENSED SOFTWARE will perform as described in the SDS manuals so long as it is operated in accordance with the written instructions of SDS. SDS' sole obligation and liability under this warranty shall be to provide corrections to the LICENSED SOFTWARE to perform as described and in no event shall SDS be liable for any special, incidental or consequential damage with regard to this warranty. In the event of a LICENSED SOFTWARE failure, SDS will supply the USER with the changes necessary to correct the errors. This warranty coverage does not include the cost of media, documentation or installation of corrections. After expiration of the six month warranty period, SDS makes no warranty of any kind.

   CAUTION: EXCEPT AS SPECIFICALLY SET FORTH HEREIN, SDS MAKES NO REPRESENTATIONS OR WARRANTIES, EITHER EXPRESS OR IMPLIED, ARISING BY LAW OR OTHERWISE, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL SDS HAVE ANY OBLIGATION OR LIABILITY ARISING FROM TORT, OR FOR LOSS OF REVENUE OR PROFIT, OR FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES.

10. SDS shall have the right to terminate the license if the USER fails to pay the license fee, or if the USER fails to comply with the license terms and conditions. The USER agrees, upon notice of such termination, to immediately return the LICENSED SOFTWARE provided by this license and all portions and copies thereof to SDS.

11. This agreement shall be governed by and interpreted pursuant to the laws of the State of North Carolina, USA. USER hereby consents to the jurisdiction of the courts of the State of North Carolina and of the United States District Court for the Eastern District of North Carolina.

sds southern data systems, inc.
5115 Holly Ridge Drive Raleigh, NC 27612 919-781-7603