

BASIC Language Reference

IBM

System/23

BASIC Language Reference

IBM

System/23

Third Edition (April, 1982)

This is a major revision of, and obsoletes SA34-0109-1. The significant changes result from information added to support the 5217 Printer and the 5247 Disk Unit.

Use this publication only for the purpose stated in the Preface.

Changes are periodically made to the information herein; any such changes will be reported in subsequent revisions or Technical Newsletters.

It is possible that this material may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Publications are not stocked at the address given below. Requests for copies of IBM publications should be made to your IBM representative or the IBM branch office serving your locality.

This publication could contain technical inaccuracies or typographical errors. A form for readers' comments is provided at the back of this publication. If the form has been removed, address your comments to IBM Corporation, Information Development, Department 27T, P. O. Box 1328, Boca Raton, Florida 33432. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

About this book

This book is a reference manual that provides specific information about the System/23 BASIC language. It was prepared with the assumption that you wish to write or change BASIC programs. This book is a precise reference which supplements *Learning System/23 BASIC*, but does not replace it.

The information in this book is in encyclopedic format and presents topics in alphabetic order. To make it easy to use, this book uses cross-references that leads you to other topics which may be of interest to you. The cross-referencing takes the following form:

Dimensioning arrays see "Declaring arrays"

This tells you that the information on dimensioning arrays can be found in the section titled "Declaring arrays."

Prerequisites

Before using this book, you should be experienced in programming the BASIC language or should have completed *Learning System/23 BASIC*.

About this book

Related publications

These books in the System/23 library contain information you may also find helpful.

- *Operator Reference*, SA34-0108
- *Learning System/23 BASIC*
 - SA34-0121—Book I
 - SA34-0122—Book II
 - SA34-0123—Book III
 - SA34-0124—Book IV
 - SA34-0125—Book V
 - SA34-0126—Book VI
 - SA34-0127—Book VII
- *System Messages*, SA34-0141
- *5110 Conversion Aid Program User's Guide*, SA34-0114
- *Customer Support Functions*, Volume 1, SA34-0175 and Volume II, SA34-0176
- *Using your 5247 Disk*, SA34-0188
- *Using your 5217 Printer*, SA34-1593

Introduction

This book contains detailed descriptions of the system commands, statements, concepts, data constants, variables, and the BASIC syntax. This is a complete reference of the BASIC language as used in System/23, and it was designed so that each topic can be found quickly. Each topic stands alone and, as in an encyclopedia, the topics are in alphabetic order by topic name.

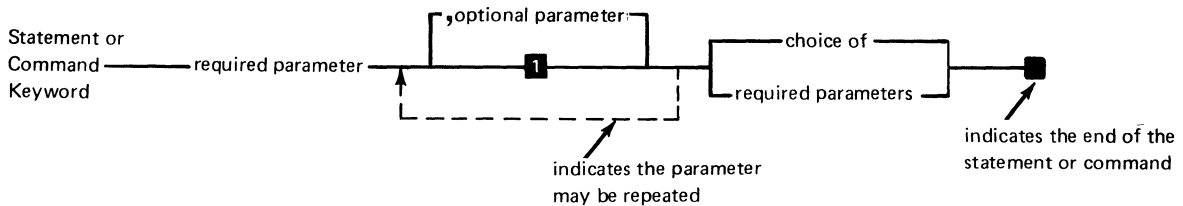
Several comprehensive programs are included in Appendix A in the back of this book. We suggest that you review these programs to get an idea of some of the ways you can use System/23 BASIC.

Syntax

Syntax description

When syntax formats are described in this manual, capitalized expressions, lowercase expressions, and special characters (such as a comma, colon, exclamation point, or an asterisk) have special meaning.

Syntax of the BASIC commands and statements in this book is presented in the following format:



Where:

Statement or Command keyword is a BASIC statement, such as LET or a command such as RUN.

required parameter is an item that must be included, such as the line reference in GOTO 100.

optional parameter is an item that may be included if desired such as ELSE in an IF, THEN, ELSE statement.

indicates that the parameter may be repeated means that more than one parameter can be included, such as the variables in INPUT A, B, C ...

choice of required parameters means that one of the parameters must be included, such as the choice between numeric or character constants in a DATA statement.

indicates the end of the statement or command refers to the block that indicates the end of the syntax.

To read the syntax of a command or statement, read from left to right along the main line. When you reach an optional parameter, you can either include that parameter or continue along the main line. When you reach a choice of required parameters, you must include one of the parameters with your command or statement.

If you do not include an optional parameter, the System/23 provides a default value or action. The defaults are listed in the description of the statement or command. The syntax diagrams include a number (such as **1**) that corresponds to the defaults listed.

In the case of the MERGE, REPLACE, and VOLID commands *only*, you must include a comma to indicate that you have omitted an optional parameter.

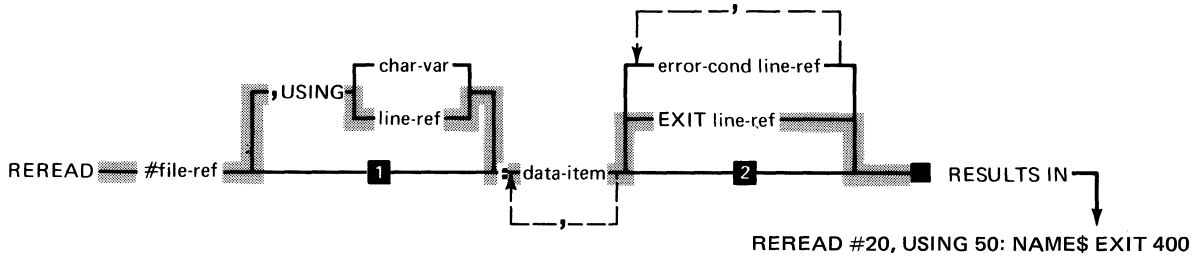
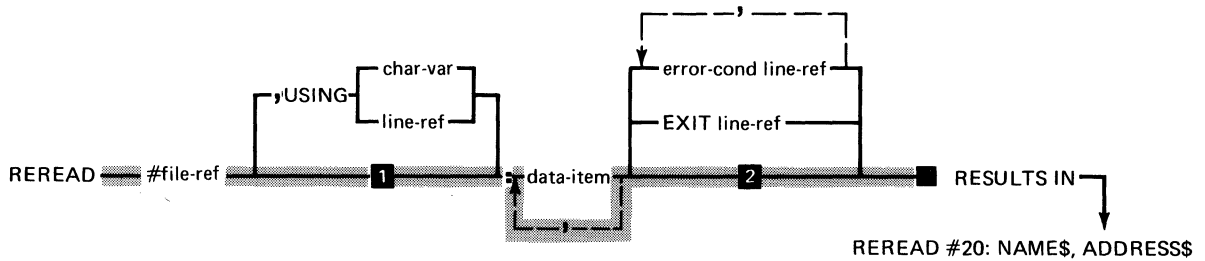
If a parameter is shown in uppercase letters, you must enter it exactly as it appears. You must also enter any special character (such as a comma or colon) that appears in the diagram.

All lines entered in program statement and commands are converted to English uppercase letters prior to syntax checking. To prevent remarks or character data on DATA statements from being converted to English uppercase, they must be enclosed in quotation marks the letters you want to remain lowercase.

Syntax

Syntax description (continued)

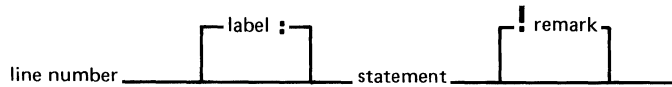
Here are two examples that use the REREAD statement:



In these examples, you must include the *file-ref* parameter following the keyword REREAD. You may choose to include the USING parameter in which case you must also include either the *char-var* or *line-ref* parameter. You must include the colon, followed by at least one *data-item*. Note that you may list more than one *data-item*. You may choose to include either *EXIT line-ref* or *error-cond line-ref*.

In the first example, the optional parameters are omitted. Therefore, the default actions are taken.

The syntax for a BASIC statement is as shown:



A keyword in a BASIC statement or system command must be followed by a blank except where a comma, parenthesis, or other appropriate delimiter is defined. Also a blank must follow the leading line number in a BASIC statement.

A label can be added to any BASIC statement except a DEF statement (see "Labels").

A remark can be added at the end of any system command or BASIC statement except a DATA statement (see "Remarks").

Because they can be used on most BASIC statements, labels and remarks are not shown in the diagrams that follow.

Absolute value

See "ABS(X)."

ABS(X)

ABS(X)

Returns the absolute value of X. The result is always positive.
For example:

```
10 X=-5.2  
20 Y=ABS(X)
```

Y contains +5.2

```
10 X=+13.7  
20 Y=ABS(X)
```

Y contains +13.7

AIDX and DIDX

The MAT assignment AIDX or DIDX statement creates an index to the elements of an array which will rearrange the original array with the elements in ascending or descending order. Character arrays are indexed alphabetically, and numeric arrays are indexed numerically.

MAT `array-name` = $\left[\begin{array}{c} \text{AIDX} \\ \text{DIDX} \end{array} \right]$ (`array-name`)

The syntax of the statement is as shown above, where:

array-name is the name of a one-dimensional array.

AIDX indicates the ascending index function.

DIDX indicates the descending index function.

When a MAT AIDX or DIDX statement is executed, index values are assigned to the array on the left of the equal sign, according to the order of the values entered into the array on the right of the equal sign.

AIDX,DIDX

AIDX and DIDX (continued)

Example (AIDX)

```
20 OPTION BASE 1
30 DIM A(10), B(10)
40 MAT B=AIDX(A)
```

If array A is		array B will be	
	9		4 (position one)
	5		9
	6		5
	0 (position four)		10
	2		8
	7		2
	8		3
	4		6
	1		7
	3		1

The numbers in array B show the position of the numbers in ascending order as they appear in array A (0 is the fourth position in array A, and it is shown as a 4 in position 1 of array B).

Example (DIDX)

```

20 OPTION BASE 1
30 DIM A(10), B(10)
40 MAT B=DIDX(A)

```

If array A is	9	array B will be	1
	5		7
	6		6
	0 (position four)		3
	2		2
	7		8
	8		10
	4		5
	1		9
	3		4 (position ten)

The numbers in array B show the position of the numbers in descending order as they appear in array A (0 is the fourth position in array A, and it is shown as a 4 in position 10 of array B).

Programming considerations

The array on the left of the equal sign must be numeric.

The results of using AIDX or DIDX depend on the collating sequence that is in effect. See "OPTION statement" and "IF, THEN, ELSE statement." See also "COLSEQ" in the *Customer Support Functions*, Volume II.

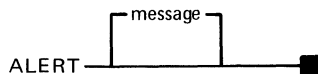
Each element of the array on the right is compared to every other element in that array to determine the index for that element. The index is then stored in the array on the left. Operation continues until the indexes for all elements of the

AIDX and DIDX (continued)

array on the right are determined. No check is made to assure that the target array is not the source array.

ALERT command

The ALERT command indicates that the operator's attention is needed during the operation of a procedure file (see "Procedure file"). Remarks are not allowed, they are interpreted as part of the message.



The syntax of the ALERT command is shown above. When executed, the ALERT command:

- Halts the system operation
- Sounds the alarm
- Displays the word ALERT and an optional message on line 22 of the display screen

The following is what the ALERT command with an optional message might look like:

```
ALERT REPLACE DISKETTE1 WITH DISKETTE2
```

The operator is informed that the diskettes must be changed.

Any command may be issued. When the command has finished processing, the keyboard is reopened for input. To continue executing the procedure file, type GO and press Enter.

To exit the procedure that issued the ALERT command enter one of the following:

- GO END
- CLEAR PROC
- PROC (for another procedure)
- CLEAR ALL

Alphabetic character set

See "Character set."

Arc tangent

See "ATN(X)."

Arithmetic arrays

Arithmetic arrays

An arithmetic array contains only numeric data and can have one or two dimensions. A one-dimensional array is a list of data items. A two-dimensional array is a matrix of rows and columns.

A(0)	B(0,0)	B(0,1)	B(0,2)	B(0,3)
A(1)	B(1,0)	B(1,1)	B(1,2)	B(1,3)
A(2)	B(2,0)	B(2,1)	B(2,2)	B(2,3)
A(3)	B(3,0)	B(3,1)	B(3,2)	B(3,3)

Note: In the above example, BASE 0 is being used. For information about BASE 0 and BASE 1, see "OPTION statement."

All elements of a numeric array (except an array received from a chaining program) are initially set to zero during the execution of the first statement that references the array.

Before being used in any of the matrix handling statements (MAT statements), an arithmetic array must be *declared* or *dimensioned*. For information on dimensioning arrays, see "DIM statement," "Declaring arrays," "Redimensioning arrays," or "MAT assignment statements."

If an array is not explicitly declared in a DIM statement, the highest subscript it can have is 10. The first reference to the array determines if the array is one- or two-dimensional.

Arithmetic data

Arithmetic data is data with a numeric value. All numbers in BASIC are decimal numbers (base 10).

Magnitude

The magnitude of a number is its absolute value. In BASIC, a power of 10 is represented by the letter E. The E is written between the first and second constant so that 10^{126} becomes 1E+126 or 1E126. 10^{-126} and 10^{126} are called floating-point numbers or notations. Floating-point notation is simply a shorthand way of expressing very large or very small numbers. See "Floating-point format" under "Arithmetic data." The numbers permitted in a BASIC program are numbers that are greater than 10^{-126} and less than 10^{126} .

Significance

The significance of a number is the number of digits it contains excluding leading and trailing zeros. For the System/23, the number is 15 digits. Numbers that are entered, are truncated to 15 digits. Numbers that are the result of an arithmetic operation, are rounded to 15 digits.

Accuracy

Additions and subtractions are accurate to 15 digits. Multiplications and divisions return 15 digits accurate to 14 digits. EXP, SQR, and exponentiation return 13 digits accurate to 12 digits. LOG, SIN, COS, TAN, and ATN return 15 digits accurate to at least 10 digits. The remaining system functions are accurate to 15 digits.

Arithmetic data

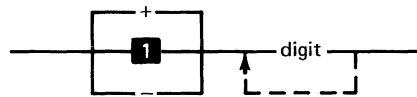
Arithmetic data (continued)

Arithmetic data formats

There are three data formats available for entering, displaying, and printing numbers: integer, fixed-point, and floating-point. Numbers in any of the formats can be positive or negative. Negative numbers must be preceded by a minus sign. Positive numbers may or may not have a plus sign.

Integer format. An integer is a whole number with no decimal point. The integer format is the same as conventional representation. A positive number may or may not be preceded by a sign.

Integer format



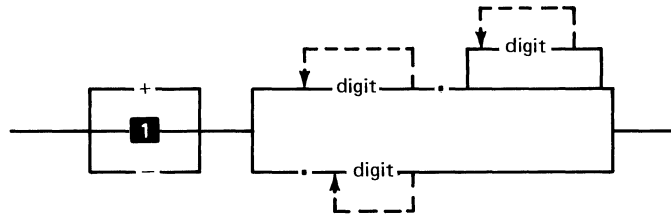
1 Positive number

Here are some examples:

0
+2
-23
266

Fixed-point format. Numbers expressed in fixed-point format are preceded by a sign and followed by a decimal point (+3.). The decimal point may also be followed by digits that express the decimal fraction (+3.56).

Fixed point format



1 Positive number

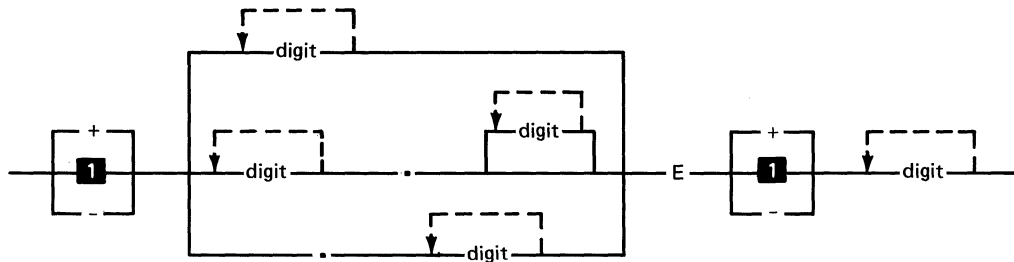
Examples of fixed-point format that are:

-.3
+3.56
33.00
33.

Arithmetic data

Arithmetic data (continued)

Floating-point format. When working with very large or very small numbers, the floating-point format is the easiest to use. Floating-point numbers are written with a fixed-point number, followed by the letter E (E stands for "multiplied by 10 to the power of"), and followed by a 1, 2, or 3-digit exponent.



1 Positive number

An example of floating-point format is:

-3.1E7

The value of the floating-point number is -3.1 multiplied by 10 to the power of 7.

-3.1E7 is the same as -3.1×10^7

-3.1E7 is the same as -31,000,000

Note that the number E7 is not a valid floating-point number. The value 10^7 must be expressed as 1E7 in BASIC floating-point format.

Selecting an arithmetic format. An arithmetic value can be entered at the keyboard in the most convenient format for the application. The number one million, for example, can be entered in any of the following ways:

```
1000000
1000000.00
1E6
+10E5
+100E+4
```

Arithmetic constant

An arithmetic constant is either an integer, a fixed-point value, or a floating-point value appearing in a BASIC statement. The value of the constant remains the same within the program. For example, the integer 1 is a constant in the statement

```
100 LET X=X+1
```


BASIC performs addition, subtraction, multiplication, division, and exponentiation. The five operators used in most formulas are:

Function	Meaning	Example
+	add, positive	$10+2=12$
-	subtract, negative	$10-2=8$
*	multiply	$10*2=20$
/	divide	$10/2=5$
** or ^	exponentiation (10 raised to the power of 2)	$10**2=100$ $10^2=100$

Rules for the arithmetic operators and the resulting actions are as follows:

Addition and multiplication: $A+B$ and $A*B$ are both commutative; or, $A+B=B+A$ and $A*B=B*A$. However, addition and multiplication are not always associative because of rounding; for example, $A*(B*C)$ does not necessarily give the same results as $(A*B)*C$.

Arithmetic expressions

Arithmetic expressions and operations (continued)

Example:

```
05 FOR I=1 to 3
10 LET A=RND
20 LET B=RND
25 LET C=RND
26 LET D=A*B*C
27 LET E=A*(B*C)
30 PRINT USING 35:D,E,D-E
35 FORM 3*N 25.17
40 NEXT I
```

Results of three typical loops:

- Contents of D
 - .04558553601442990 (first time)
 - .00548795587029670 (second time)
 - .43103396752564700 (third time)
- Contents of E
 - .04558553601443000 (first time)
 - .00548795587029669 (second time)
 - .43103396752564700 (third time)
- Difference (D-E)
 - .00000000000000010 (first time)
 - .00000000000000001 (second time)
 - .00000000000000000 (third time)

Subtraction: $A-B$ is defined as A minus B.

Division: A/B is defined as A divided by B. If $B=0$ and A is not 0, an error (zero divide) occurs. If $A=0$ and $B=0$ the result is 1.

Exponentiation: The expression $A**B$ or $A \wedge B$ is defined as the value of the variable A raised to the B power. The following rules apply to exponentiation:

- If $A=0$ and $B<0$, a zero divide error is returned
- If $A<0$ and B is not an integer, an error occurs because of a negative number to a fractional power
- If $B=0$, $A**B$ equals 1
- If $A=0$ and $B>0$, $A**B$ equals 0

Considerations:

- Exponentiation returns 13 digits accurate to 12.
- The circumflex “ \wedge ” can also be used for exponentiation; however, the system converts the circumflex to $**$. The circumflex key on the keyboard does not advance the cursor.

Arithmetic expressions

Arithmetic expressions and operations (continued)

Positive/negative operations: The + and – signs can also be used as positive/negative operators. These positive/negative operators can be used in only two situations. They are:

- Following a left parenthesis and preceding an arithmetic expression
- As the leftmost character in an entire arithmetic expression

For example:

$-A+(-B)$ and $B-(-2)$ are valid

$A+-B$ and $B--2$ are invalid

For more information on arithmetic expressions and operations, see "Arithmetic hierarchy."

Subjects related to arithmetic expressions

ABS	DISPLY	LINE	RND
AIDX	ERR	LOG	ROUND
ATN	EXP	MAX	SGN
CEIL	FILE	MIN	SIN
CMDKEY	FILENUM	ORD	SQR
CNT	FREESP	PI	SRCH
CODE	INT	POS	TAN
CON	KLN	PROCIN	UDIM
COS	KPS	REC	VAL
DIDX	LEN	RLN	ZER

Arithmetic hierarchy

Expressions with two or more operations are performed according to the hierarchy of the operations involved. BASIC performs the operations in the following order:

1. Parentheses receive top priority. When parentheses are nested (within another set of parentheses), the operation in the innermost pair is performed first.
2. If there are no parentheses, the order of priority is:
 - a. Exponentiation (^ or **).
 - b. Positive and negative.
 - c. Multiplication (*) and division (/) have equal priority.
 - d. Addition (+) and subtraction (-) have equal priority.
3. If the items are of equal priority, then the evaluation of the operators is from left to right. The following are examples of arithmetic hierarchy, showing how expressions are evaluated:

- Parentheses ()
 $70 - (25 + 15) = 70 - 40 = 30$
- Exponentiation **
 $10 + 10^{**}2 = 10 + 100 = 110$
- Multiplication * or division /
 $10 + 10 * 2 = 10 + 20 = 30$
 $10 + 10 / 2 = 10 + 5 = 15$
 $10 + 10 * 2 / 5 = 10 + 20 / 5 = 10 + 4 = 14$
- Addition + or subtraction -
 $10 + 10 = 20$
 $10 - 5 = 5$
 $10 + 10 - 5 = 20 - 5 = 15$

Arithmetic hierarchy

Arithmetic hierarchy (continued)

- Nested parenthesis
 $150/(2*(13 + 12)) = 150/(2*25) = 150/50 = 3$

The entire hierarchy would be as described below:

In Step 1, the nested parenthesis $(13 + 12)$ is performed.

Step 1. $50 + 10^{**}2/(2*(13 + 12)) - 2 =$

In Step 2, the parenthesis $(2*25)$ is performed.

Step 2. $50 + 10^{**}2/(2*25) - 2 =$

In Step 3, the exponentiation $10^{**}2$ is performed.

Step 3. $50 + 10^{**}2/50 - 2 =$

In Step 4, the division $100/50$ is performed.

Step 4. $50 + 100/50 - 2 =$

In Step 5, because addition and subtraction have equal priority, the priority is from left to right. The addition $50+2$ is performed.

Step 5. $50 + 2 - 2 =$

In Step 6, the final step, the subtraction $52-2$ is performed and the answer is shown.

Step 6. $52 - 2 = 50$

See "Arithmetic expressions and operations."

Arithmetic variables

A variable represents a number whose value is subject to change during the execution of the program. Arithmetic variables have names consisting of from one to eight alphabetic or numeric characters, with the first being alphabetic. Examples of valid variable names are:

```
A5  
BASIC  
DATA1  
BYTE12
```

Arithmetic variables are stored internally as decimal floating point.

An arithmetic variable is initially set to zero during the execution of the first statement that references the variable (except when the variable is received from a chaining program).

Some names are reserved by the system and cannot be used for variables or labels. See "Reserved words." The term variable includes array elements (see "Arithmetic arrays").

Array expressions

See "MAT assignment statements."

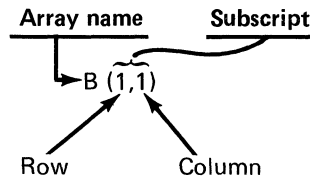
Arrays

Arrays

An array is a collection of data items (elements) that is referred to by a single name. Only data items of the same type (numeric or character) can be grouped together to form an array. An array is a convenient tool that provides a fast and organized way of handling large amounts of data within a program.

Arrays can be either one- or two-dimensional. A one-dimensional array can be thought of as a list of successive data items. A two-dimensional array can be thought of as a matrix of rows and columns.

Each element in an array is referred to by the name of the array followed by a subscript enclosed in parenthesis. Array subscripts can begin with either zero (BASE 0 indexing) or one (BASE 1 indexing). You can select the base by using the OPTION statement (see "OPTION statement"). The default is BASE 0.



OPTION BASE 0 indicates that the first element of an array has a subscript of 0.

OPTION BASE 1 indicates that the first element of an array has a subscript of 1.

There are two types of subscripts. One is specified by a single number after the array name, such as A(3).

For example:

```
10 OPTION BASE 1
20 DIM A(3) !DEFINES A ONE-DIMENSIONAL ARRAY
.
.
.
60 LET A(3)=468.45 !REFERENCES THIRD ELEMENT
```

A(0)
A(1)
A(2)
A(3)

Base 0
(default)

A(1)
A(2)
A(3)

Base 1

Note: If OPTION BASE 1 was not specified, then statement 60 would be referencing the fourth element.

Arrays

Arrays (continued)

The other type of subscript has two numbers after the variable name, such as B(3,3).

For example:

```
10 OPTION BASE 1
20 DIM B(3,3) !DEFINES TWO-DIMENSIONAL ARRAY
30 LET B(2,2)=9.6 !REFERENCES ROW 2, COLUMN 2
```

B(0,0)	B(0,1)	B(0,2)	B(0,3)
B(1,0)	B(1,1)	B(1,2)	B(1,3)
B(2,0)	B(2,1)	B(2,2)	B(2,3)
B(3,0)	B(3,1)	B(3,2)	B(3,3)

Base 0
(default)

B(1,1)	B(1,2)	B(1,3)
B(2,1)	B(2,2)	B(2,3)
B(3,1)	B(3,2)	B(3,3)

Base 1

When referencing an element in a one-dimensional array, the position of the element is obtained by counting from top to bottom. Thus, assuming BASE 0, the fourth element of a one-dimensional array named A can be referenced by the symbol:

A(3)

The first value in a subscript of a two-dimensional array gives the number of the row containing the referenced element. Rows are numbered from top to bottom. The second value in the subscript gives the number of the column. Columns are numbered from left to right. Thus, assuming BASE 0, the third element in the fifth row of a two-dimensional array named B can be referenced by the symbol:

B(4,2)

Each subscript value can also be an arithmetic expression. For example, if I=3 then row 5, column 3 of the array named B can be referenced by the symbol:

B(I+1,2)

The maximum subscript is 9999.

See "Program 1—Sample" in Appendix A.

Arrays, arithmetic

See "Arithmetic arrays."

Arrays

Arrays, character

See "Character arrays."

Arrays, declaring

See "Declaring arrays" under "DIM statement."

Arrays, redimensioning

See "Redimensioning arrays."

Ascending index

See "AIDX and DIDX."

Assignment statements

See "LET statement."

ATN(X)

Returns the arc tangent of X, where X is in radians.

Attention and inquiry

A BASIC program may be interrupted by the operator in one of two ways:

- Cmd/Attn (press and hold the Cmd key, then press the Attn key)
- Inq (Inquiry) key.

Cmd/Attn can be used at any time and will stop the execution of a BASIC program following the statement during which it is pressed. The System/23 goes into "split screen mode."

If the Cmd/Attn is detected during a user defined function, the program gets an error indicating that a user function was interrupted. The operator resumes normal execution with Error Reset or may terminate the user function with Cmd/Error Reset which abandons execution of the function. (GO will continue execution following the line that invoked the function.) See "DEF, FNEND statement."

If a procedure is running, the procedure is interrupted following the command being executed.

If Cmd/Attn is pressed during the execution of a RUN command in a procedure, the program interrupts as if a procedure does not exist. To interrupt the procedure when the program ends, end the program with a PAUSE statement; then, enter commands and restart the procedure.

The commands LOAD, SAVE, and REPLACE cannot be interrupted.

Attention

Attention and inquiry (continued)

During an interrupt, any system commands or calculator statements can be entered. Some statements will prevent resumption of the interrupted program or procedure (for example, CLEAR, LOAD, and LINK).

Some statements will be rejected if their execution would cause ambiguous results (editing OPTION, DIM, FOR, or NEXT statements).

Normal execution can be resumed by entering GO.

No error code is set by Cmd/Attn.

The Inq key also interrupts a BASIC program. The response to the Inq key is controlled by the ON statement (see "ON statement"). The default action is to interrupt execution with a 0001 error. The ON statement may also specify that the Inq key be IGNORED or cause a GOTO when it is pressed. The inquiry key is ignored during execution of a GOTO statement. It is not advisable to execute a one-statement loop (10 GOTO 10) while waiting for the Inq key to be pressed. The CONTINUE statement may be used to return control to the interrupted task (see "CONTINUE statement").

The Inq key is not checked while a defined function is executing. It is processed normally after all defined functions are completely executed.

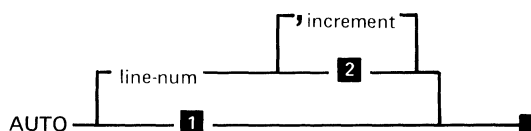
Inadvertently pressing the HOLD or TEST keys may result in entering system diagnostic mode.

See "Program 1—Sample" in Appendix A.

AUTO command

The AUTO command numbers program lines or DATA statements. The starting line number and the increment can be specified. If a beginning line number or increment value is not specified, a beginning line number of 10 and an increment of 10 is generated for BASIC programs or data statements.

Be sure that the AUTO command does not replace existing lines you do not want changed.



- 1** AUTO 10, 10
- 2** AUTO line-num,10

The syntax of the AUTO command is as shown where:

line-num is a positive number specifying the first line number to be generated. The range of this number is from 1 to 99999. The default is 10.

increment is a positive integer from 1 to 99998 used to increment succeeding line numbers. If a beginning line number is not specified, the increment cannot be specified. The default is 10.

Each line number generated by the AUTO command for a BASIC program is followed by a blank, then the cursor.

00010 _

AUTO

AUTO command (continued)

When working with a data file, the line number is followed by a colon and then the cursor.

00010: __

Examples: To use line numbers

AUTO 10,20,30,40,etc.

AUTO 15 15,25,35,45,etc.

AUTO 15,5 15,20,25,30,etc.

AUTO 150,25 150,175,200,225,etc.

Programming considerations

- Ending AUTO
 - Automatic line numbering continues until the line number put on the screen is overwritten or until an empty line is scrolled up.
- Procedures
 - Automatic line numbering cannot be done from a procedure file.
- Entering DATA
 - If AUTO is used to enter data, CLEAR DATA or LOAD...,DATA must be issued first.
- Adding lines
 - To find the last line for continued entry: LIST 99999

Be sure that the AUTO command does not replace existing lines that you do not want changed.

BASIC statements

A BASIC program is made up of BASIC statements. BASIC statements allow you to enter data, specify how that data is to be used, and determine the output. BASIC statements are either executable or descriptive (nonexecutable). Executable statements cause a program action such as value assignment or printing. Descriptive statements provide information needed by the program or the user, but they do not cause a visible action.

BASIC statements can be up to 255 characters including six for the line number and following blank. The maximum number of statements permitted in a single BASIC program is limited by the work area size of the system, the statement types, and the maximum line number (99999).

The statements and a brief description are listed here.

CHAIN	Ends a program, then loads and begins executing another program or a procedure.
CLOSE	Closes a file that is open.
CONTINUE	Transfers control to the statement following the one causing the ON-condition transfer or I/O exit.
DATA	Creates an internal data table of values.
DEF	Defines a function to be used in the program.
DELETE	Marks a specific record in an internal I/O file as unavailable (deleted).
DIM	Specifies the size of an array or character variable length.
END	Ends a program.
EXIT	Specifies error exits for corresponding error conditions.

BASIC

BASIC statements (continued)

FNEND	Ends a function defined in a DEF statement.
FOR	Begins a loop and determines when the loop is exited (as used with a NEXT statement).
FORM	Specifies format for displayed/printed input/output and records in files.
GOSUB	Transfers control to the beginning of a subroutine.
GOTO	Transfers control to a specific statement.
IF,THEN,ELSE	Transfers program control or executes a statement according to the results of the logical expression.
INPUT	Assigns values from the keyboard or other device to variables. or array elements during program execution.
LET	Assigns values to variables.
LINPUT	Performs unformatted character string input.
MAT	Assign values to all elements of an array.
NEXT	Last statement in a loop (see FOR).
ON	Specifies a transfer of control when certain events take place.
OPEN	Activates internal or display files for input or output.
OPTION	Set global parameters of BASIC program.
PAUSE	Halts program execution.
PRINT	Transfers DISPLAY data to a specified device.
RANDOMIZE	Sets a new starting point in random number generator.

READ	Assigns values from the internal table (see DATA) or internal I/O files to variables or array elements.
RELEASE	Removes the lock on a disk file so that other computers can use the file.
REM	Defines comments or remarks in a program.
REREAD	Allows access to the last record obtained from a file.
RESERVE	Locks an entire file against access by any other computer. Applies only to files on the 5247 Disk.
RESTORE	Causes values in the internal data table (see DATA) to be assigned starting with the first table value, resets the data file to the beginning or to a specific record.
RETRY	Transfers control to the statement causing the most recent error.
RETURN	Ends a current subroutine.
REWRITE	Updates existing record in a file.
STOP	Stops execution of program statements.
TRACE	Traces all or part of a program's execution.
USE	Defines the names of the variables passed by the CHAIN statement.
WRITE	Adds a record to an internal I/O file.

Note: More information on individual statements may be found by locating the statements, listed in alphabetic order, in this book.

Blanks

Blanks

The following rules apply to the use of blanks:

- Blanks can be used within quoted character strings.
- A blank or other syntactically defined delimiter is required after a keyword.
- Blanks are not allowed within keywords, variable names, numeric constants, function names, line numbers, and labels.
- Nonsignificant blanks are deleted when the program is listed (see "LIST, LISTP command").
- Blanks are required after leading line numbers in BASIC statements.
- To retain blanks, the program must be entered/edited in DATA mode.
- Blanks are significant in relational compares.

Throughout this book the symbol $\text{\textcircled{b}}$ will represent a blank.

Byte

The unit of machine and diskette or disk storage. For example, one character takes one byte.

Catenation

See "Concatenation."

CEIL(X)

Returns the smallest whole number (integer) greater than or equal to X. For example:

```
10 CEIL (-1.2)=-1  
20 CEIL (+2.3)=3
```

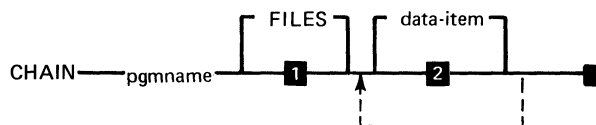
Ceiling

See "CEIL(X)."

CHAIN

CHAIN statement

The CHAIN statement ends the program being executed, loads another program, and starts executing the new program. The CHAIN statement may also be used to start a procedure or a subprocedure from a BASIC program.



- 1** All files are closed
- 2** No data is passed

The syntax of the CHAIN statement is as shown above, where:

pgmname is a character expression representing the program name (see "File specifications"). If the first five characters of *pgmname* are PROC=, then the file is invoked as a procedure. If the first eight characters are SUBPROC=, then the file is invoked as a subprocedure. In either case, *FILES* and *data-item* may not be specified. The file specification parameter must be used to address files and PROCs located on the 5247 Disk Unit.

FILES indicates that all files of the current program remain open and at their current positions. If the keyword *FILES* is not specified, all files except procedure files are closed when the CHAIN statement is executed.

data-item is the name of a variable or array (without the keyword MAT).

The data items define the names of the variables that are to retain their data when the chain occurs. All other variables are destroyed during the chaining operation. The syntax of the list of data items is not checked until the CHAIN statement is executed.

Examples

In the following example, the current program ends all files are kept open, PGM3 from VOL1 is loaded, and the values of variables A and B\$ are copied into the chained-to program.

```
10 CHAIN "PGM3/VOL1," FILES, A, B$
```

In the following example, the system chains to the procedure file "PROC4." In statement 70, "PROC=" is necessary to indicate that "PROC4" is a procedure and not a program.

```
70 B$ = PROC=PROC4  
80 CHAIN B$
```

If a procedure is already in effect, it is replaced with the new procedure, PROC4.

An example of CHAIN specifying the subprocedure is:

```
90 CHAIN "SUBPROC=SET.TIME"
```

If a procedure is already in effect, it resumes control when the subprocedure is finished.

CHAIN

CHAIN statement (continued)

Programming considerations

- USE
 - The chained program must contain a USE statement that specifies the same variables in the same order as the CHAIN statement (see “USE statement”).
- Dimensioning
 - The chaining and the chained program must be dimensioned the same way as all arrays and the character variables that are passed. The programs can redimension the arrays in any valid manner (see “Redimensioning arrays”).
- Options
 - The options specified on an OPTION statement in the chained-from program must match the options specified in the chained-to program.
- IF, THEN, ELSE
 - There cannot be an ELSE clause when a MAT or CHAIN statement is the object of a THEN clause. Only a remark can follow the data items in a CHAIN statement.
- CHAIN interrupt
 - If CHAIN processing gets interrupted for any reason while LOAD appears on the status line (for example, file not found) and if termination is desired, CLEAR ALL must be entered.

See “Program 7—Sample” in Appendix A.

Character arrays

A character array contains only character data and can have one or two dimensions.

Character arrays, like simple character variables, are named by a single letter of the alphabet followed by zero to seven alphabetic or numeric characters, followed by the dollar sign (\$).

For example:

```
D$(5) = "JONES"  
A5$(10) = "SMITH"
```

Character arrays can be used in input, output, and simple matrix assignment statements. They can be redimensioned (except for maximum string length). The maximum string length of each element of a character array cannot be changed.

For more information, see:

“Arrays”

“Character variables”

“Redimensioning arrays”

“DIM statement”

“VAL(A\$)”

Character constants

Character constants

A character constant is a string of characters enclosed in quotation marks. Any letter, digit, or special character can be in a character constant. For example "THE PRICE IS \$6.95." represents THE PRICE IS \$6.95.

The character constant, including blanks but excluding the delimiting quotation marks, may be from zero through 255 characters long. The following are examples of valid character constants:

```
"YES"  
"HE SAID ""HELLO"""  
"123456"
```

Lowercase characters within quotes (constants) are not changed to uppercase.

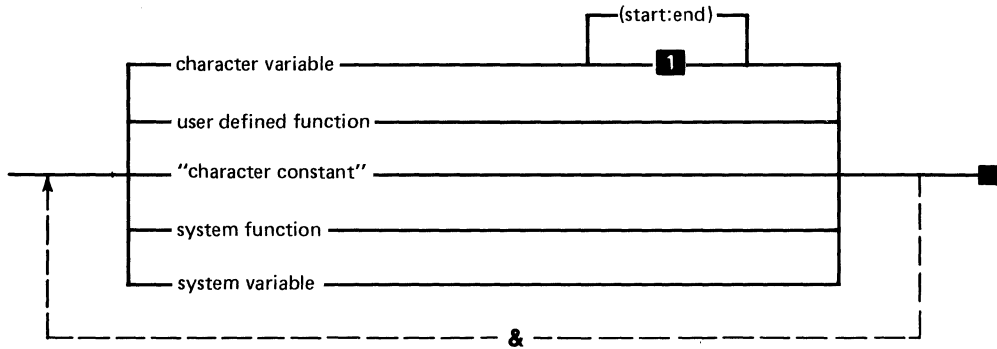
To represent quotes within character strings, two consecutive quotes (" ") are required.

Character data

Character data in BASIC is data with a character value. It can be in the form of constants or variables (see "Character constants" and "Character variables").

Character expressions

Syntax



1 Entire variable

Start and End are numeric expressions.

A character expression is a character constant, a character variable, a character operation reference, a single element of a character array, a character substring, or a combination of these. The only operators ever associated with character expressions are the substring and the concatenation symbol. For more information, see "Concatenation" and "Substring referencing." The following are examples of character expressions:

```
"ABCDEFGH123456"
ALPHA$ & BETA$
"SER" & "IAL"
ZEBRA$(2:6)
```

Character expressions

Character expressions (continued)

Subjects related to character expressions

This section lists and summarizes subjects related to character expressions. For additional information, refer to the specific subject in this manual.

Character set	See charts under this topic.
CHR\$	Returns character for specified position within collating sequence.
Concatenation	Joins character strings together.
DATE\$	Returns date set by DATE command.
FILE\$	Returns file specification.
FORM statement	Specifies format for displayed/printed input/output and for records in files.
HEX\$	Returns hexadecimal value.
KSTAT\$	Returns the most recent keystroke.
LEN	Returns the length of a string.
LPAD\$	Returns a string padded on the left with blanks.
LTRM\$	Returns a string with leading blanks removed.
ORD	Returns ordinal value.
PIC\$	Returns/changes the current currency symbol.
POS	Returns position of matching substring.
RPAD\$	Returns a string padded on the right with blanks.
RPT\$	Returns repeated character.
RTRM\$	Removes trailing blanks.
SRCH	Searches array for a value.

Character expressions

SREP\$	Replaces strings past a specified position with another string.
STR\$	Converts a specified value to a character string.
TIME\$	Returns time of day (set initially by TIME command).
WSID\$	Identifies which connector number on the 5246 Diskette Unit or on the 5247 Disk Unit that the computer is attached to.

Character set

Character set

The System/23 character set is used to represent arithmetic and character data as data constants and variables.

The character set consists of the following:

- Alphabetic characters (English)
- Alphabetic characters (non-English)
- Numeric characters
- Special characters
- Graphic characters

Alphabetic characters (English)

The uppercase and lowercase letters of the alphabet (A through Z) make up the System/23 alphabetic characters.

Alphabetic characters (non-English)

Characters of the alphabet that are non-English may not be used for BASIC variable names and filenames.

Numeric characters

In BASIC, the numeric characters are the digits 0 through 9.

Special characters

There are 21 characters that have special meaning in System/23 BASIC:

Character	Name
	Blank or space
=	Equal sign
+	Plus sign
-	Minus sign
*	Asterisk
/	Slash
^	Circumflex
(Left parenthesis
)	Right parenthesis
,	Comma
.	Period or decimal point
;	Semicolon
:	Colon
&	Ampersand, concatenation
?	Question mark
>	Greater than
<	Less than
!	Exclamation point
\$	Dollar sign
"	Quotation mark
#	Number sign

Character set

Character set (continued)

The cursor does not move to the right when using the circumflex. The Cursor Right key must be used.

There are other special characters but they do not have special meaning in System/23 BASIC. They are used within character strings.

Graphic characters

There are 11 graphic characters in System/23 BASIC:

- Vertical bar |
- Lower right corner }
}
- Lower tee T
- Left tee F
- Upper tee T
- Upper left corner L
- Lower left corner L
- Upper right corner L
- Right tee T
- Horizontal bar -
- Intersection +

Note: On the printers, small gaps may be visible between graphic characters.

Character set

The Display attributes, highlight and blink, do not affect these graphic characters.

The following chart lists all of the EBCDIC characters and their hexadecimal representation used in System/23 BASIC.

Column	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
Row ↓ Bit Pat. →	00				01				10				11				
	00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11	
0	0000			┘	┐	SP	&	-	φ	φ	°	μ	ç	{	}	\	0
1	0001			└	┌	RSP	é	/	É	a	j	~	£	A	J	NSP	1
2	0010			┌		â	ê	Â	Ê	b	k	s	¥	B	K	S	2
3	0011			U		ä	ë	Ä	Ë	c	l	t	Pt	C	L	T	3
4	0100	H	Norm	ND	—	à	è	À	È	d	m	u	f	D	M	U	4
5	0101				RHB	á	í	Á	Í	e	n	v	§	E	N	V	5
6	0110					ã	î	Ã	Î	f	o	w	¶	F	O	W	6
7	0111	B	UR	RH		â	ï	Â	Ï	g	p	x	¼	G	P	X	7
8	1000	R	URH	T		ç	ì	Ç	Ì	h	q	y	½	H	Q	Y	8
9	1001		URB	Γ		ñ	β	Ñ	`	i	r	z	¾	I	R	Z	9
A	1010	HB	URBH			[]		:	«	á	í	—	SHY	ı	²	³
B	1011	RB				.	\$,	#	»	ó	ı	†	ô	û	Ô	Û
C	1100		UH	L		<	*	%	@	đ	æ	ð	-	ö	ü	Ö	Ü
D	1101		UB			()	_	'	ý	,	ý	..	ò	ù	Ò	Ù
E	1110		UBH		+	+	;	>	=	‡	Æ	İ	'	ó	ú	Ó	Ú
F	1111					!	^	?	"	±	⊗	®	=	õ	ÿ	Õ	EO

Character set

Character set (continued)

Special use characters

The following shows characters that perform a special function on the display screen and/or printer. An X indicates that the device supports the function.

Character set

Hex	ID	Screen	Printer	Use
04	H	X	-	Highlight
06	New line	Blank	X	Start output in column 1, next line
07	B	X	-	Blink
08	R	X	-	Reverse image
0A	HB	X	-	Highlight blink
0B	RB	X	-	Reverse image,blink
0C	New Page	X	X	Screen=clear, printer=eject page
0D	CR	X	X	Carrier return
11		Blank	X	System use only
12		Blank	X	System use only
13		Blank	X	System use only
14	N	X	X	Screen=normal image,no blink,highlight,underline Printer=stop underline
15	New line	X	X	Start output in column 1,next line
17	UR	X	-	Underline,reverse image
18	URH	X	-	Underline,reverse image,highlight
19	URB	X	-	Underline,reverse image,blink
1A	URBH	X	-	Underline,reverse, blink,highlight
1C	UH	X	-	Underline,highlight
1D	UB	X	-	Underline,blink
1E	UBH	X	-	Underline,blink, highlight
23	U	X	X	Underline
24	I	X	-	Invisible
25	Line feed	Blank	X	Start output in same column,next line
27	RH	X	-	Reverse,highlight
2B	Format	Blank	X	Set printer
35	RHB	X	-	Reverse image, highlight,blink
3A	Page End	Blank	X	Eject page
3B		Blank	X	System use only
3C		Blank	X	System use only

Character set

Character set (continued)

Characters not displayable

Not all graphics can be displayed at the same time (see "DISPLY"). Characters that cannot be displayed show up as "blobs." All graphics always print on the printer (except 1/4, 1/2, and 3/4 on some printers).

The following charts show which characters are not displayed for each setting of DISPLY. Characters that cannot be displayed are shown in the shaded boxes.

DISPLY(1) — United States character set:																	
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0						&	-	ø	°	μ	Ç	{	}	\	0		
1						é	/		a	j	-	£	A	J	1		
2									b	k	s	¥	B	K	S	2	
3									c	l	t	Pts	C	L	T	3	
4									d	m	u	f	D	M	U	4	
5						á	í		e	n	v	§	E	N	V	5	
6						ä	Ä		f	o	w	¶	F	O	W	6	
7						ï			g	p	x		G	P	X	7	
8						ç	Ç	ì	h	q	y		H	Q	Y	8	
9						ñ	β	Ñ	`	i	r	z		I	R	Z	9
A						[]	ı	:	«	»	ı	ı	ı	ı	ı	ı
B						•	\$,	#	»	ı	ı	ı				
C						<	*	⌘	@					ü			
D						()	-	'	ı				ö			
E						+	;	>	=					ó	ú		
F						!	^	?	"	±		®	=	ò			õ

DISPLY(2) — Canada character set:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
0					&	-	°	μ	ç	{	}	\		0				
1					é	/	É	a	j	~	£	A	J			1		
2					â	ê	À	Ê	b	k	s	¥	B	K	S	2		
3					ä	ë	Ä	Ë	c	l	t	■	C	L	T	3		
4					à	è	À	È	d	m	u	f	D	M	U	4		
5									e	n	v	§	E	N	V	5		
6									ï	í	f	o	w	■	F	O	W	6
7									ï	ĩ	g	p	x	■	G	P	X	7
8									ç	ç	ì	h	q	y	H	Q	Y	8
9									·	·	i	r	z	■	I	R	Z	9
A					[]	!	:	«	»	—	—	—	—	1	2	3	
B					·	\$,	#	»	»	†	ø	ú	ô	ù			
C					<	*	æ	@	■	■	■	■	ü	■	Ü			
D					()	-	'	■	■	■	■	ù	■	Ù			
E					+	;	>	=	■	■	■	■	·	■	·			
F						^	?	"	■	■	■	■	=	■	ÿ	■		

DISPLY(3) — Europe (except Spain) character set:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
0					&	-	ø	φ	°	μ	■	{	}	\		0		
1					é	/	É	a	j	~	£	A	J			1		
2					â	ê	■	■	b	k	s	¥	B	K	S	2		
3					ä	ë	Ä	■	c	l	t	■	C	L	T	3		
4					à	è	■	■	d	m	u	f	D	M	U	4		
5									e	n	v	§	E	N	V	5		
6									ï	í	f	o	w	■	F	O	W	6
7					á	ã	À	■	g	p	x	■	G	P	X	7		
8					ç	ï	■	■	ì	h	q	y	H	Q	Y	8		
9					■	β	■	■	·	·	i	r	z	■	I	R	Z	9
A					[]	!	:	«	»	—	—	—	—	1	2	3	
B					·	\$,	#	»	»	†	ø	ú	■				
C					<	*	æ	@	■	■	■	■	ü	■	Ö	Ü		
D					()	-	'	■	■	■	■	ù	■	Ò	■		
E					+	;	>	=	■	■	■	■	·	■	·			
F						^	?	"	■	■	■	■	=	■	ÿ	■		

Character set

Character set (continued)

DISPLY(4) — Nordic (including Iceland) character set:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0					&	-	ø	°		{	}	\				0	
1				é	/	É	a	j	-	£	A	J				1	
2									b	k	s	ŷ	B	K	S	2	
3				ä		Ä	c	l	t				C	L	T	3	
4									d	m	u	f	D	M	U	4	
5				á	í	Á	í	e	n	v	§		E	N	V	5	
6									f	o	w		F	O	W	6	
7				ð	ī	Ā			g	p	x		G	P	X	7	
8									h	q	y		H	Q	Y	8	
9									β							9	
A				[]		:								1	2	3
B				.	\$,	#										
C				<	*	%	@	đ	æ	ð	-	ö	ü	ö	ü		
D				()	-	'	ý	s	ý	"				ò		
E				+	;	>	=	ı	Æ	I	'	ó	ú	ó	ú		
F				!	^	?	"										

DISPLY(5) — Spain (Spanish speaking) character set:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0					&	-					ç	{	}	\		0	
1					é	/			a	j	-	£	A	J		1	
2				â	ê				b	k	s	ŷ	B	K	S	2	
3				ã	ë	Ä			c	l	t			C	L	T	3
4				à	è				d	m	u	f	D	M	U	4	
5				á	í				e	n	v			E	N	V	5
6					ı				f	o	w			F	O	W	6
7					ı				g	p	x			G	P	X	7
8				ç	ı	Ç	ı		h	q	y			H	Q	Y	8
9				ñ	β	Ñ	`		i	r	z			I	R	Z	9
A				[]		:		g	ı					1	2	3
B				.	\$,	#		g	ı		ò	ú				
C				<	*	%	@					ö	ü	ö	ü		
D				()	-	'		s		"	à	ù	ò			
E				+	;	>	=				'	ó	ú				
F				!	^	?	"										

Character set

	0	1	2	3	4	5	6	7	8	9
0										
1			Note 2					Note 1		
2								ı		
3			ı	ı	ı					
4	ı	ı		Note 1	ı				ı	ı
5			—							
6			+				â	ä	à	á
7	ã	ä	ç	ñ	[.	<	(+	!
8	&	é	ê	ë	è	í	î	ï	ì	β
9]	\$	*)	;	^	-	/	Â	Ä
10	À	Á	Ã	Ä	Ç	Ñ		,	%	_
11	>	?	φ	É	Ê	Ë	È	Í	Î	Ï
12	Ï	`	:	#	@	'	=	"	φ	a
13	b	c	d	e	f	g	h	i	≤	≥
14	đ	ý	ı	±	·	j	k	l	m	n
15	o	p	q	r	ä	ö	æ	ş	Æ	☉
16	μ	~	s	t	u	v	w	x	y	z
17	ı	ı	Đ	Ÿ	ı	®	ç	£	¥	ı
18	f	ğ	ı	¼	½	¾	ı		ı	ı
19	ı	=	}	A	B	C	D	E	F	G
20	H	I	ı	ô	ö	ò	ó	õ	}	J
21	K	L	M	N	O	P	Q	R	ı	û
22	ü	ù	ú	ÿ	\		S	T	U	V
23	W	X	Y	Z	²	ô	ö	ò	ó	õ
24	0	1	2	3	4	5	6	7	8	9
25	³	û	ü	ù	ú					

Notes:

1. Unprintable character
2. Page advance

Use table from left to right

Examples:

decimal code 193 prints **A**
 decimal code 91 prints **\$**

The information in this table is used with **CHR\$**

Decimal representation of characters

Character variables

Character variables

A character variable is a named item of character data whose value is subject to change during program execution. Character variables are named by a single letter of the alphabet, followed by from zero to seven alphabetic or numeric characters, followed by the dollar sign \$.

When the program is executed, the initial value of character variables is set to null (zero length).

When a character expression value is assigned to a character variable, the resulting length of the character variable is that of the expression.

For example:

```
A$="ABC"
```

A\$ is now three characters long.

The maximum length a character variable can be dimensioned to is 255 characters. Examples of character variables are:

```
A$  
DATA$  
NAME$  
M211$
```

The maximum length of a character variable is 18 unless specified differently in a DIM statement (see "DIM statement").

CHR\$(X)

Returns the one-character string occupying the ordinal position X within the native System/23 collating sequence. X must be in the decimal range from 0 through 255. If X is outside this range, an error occurs. The change collating sequence Customer Support Function does not affect the result of CHR\$. For additional information, see "Character set."

Example:

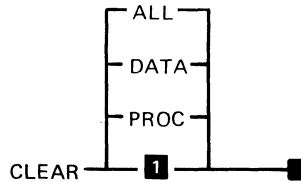
```
10 X=247
20 A$=CHR$(X)
```

A\$ contains "7"

CLEAR

CLEAR command

The CLEAR command deletes the program or data file from the work area, or cancels the active procedure(s).



1 CLEAR to PROGRAM mode.

The syntax for the CLEAR command is as shown above, where:

ALL clears the work area of the program or data and any active procedure files. In effect, it puts the machine in an initial power-on status. This reinitializes all system variables except *TIME\$* and *WSID\$*.

DATA sets the work area to DATA mode for entering data files on the keyboard. Data closes all files (except procedure files) left open by the program in the work area. All contents of the work area are erased.

PROC resets the system to keyboard input and eliminates any active procedure file hierarchy. Closes all procedure files left open. The contents of the work area are not erased.

If no parameter is specified, *PROGRAM* is the default. It sets the work area to PROGRAM mode for program entry. This closes all files (except procedure files) left open by the program in the work area. All contents of the work area are erased.

The status line will display READY INPUT when the command is complete.

CLEAR should be used whenever a new program is entered. Otherwise, existing lines in the work area may become part of the new program.

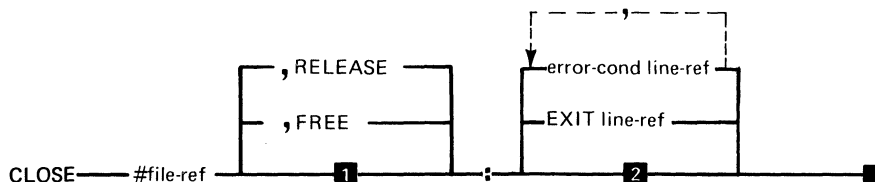
Clear display screen

See "NEWPAGE" under "Print statement."

CLOSE

CLOSE statement

The CLOSE statement specifies the file to be closed. CLOSE is executed for each active file at the end of program execution.



- 1 Keep file, maintain reserve status
- 2 Interrupt on error unless ON is active

The syntax of the CLOSE statement is shown above, where:

file-ref is a numeric expression. See "File-reference parameter."

RELEASE resets any reserve control status. See "File sharing."

FREE frees the file if it is opened NOSHR. See "DROP/FREE command."

error-cond line-ref specifies the line number or label that the program should transfer to for one of the following error conditions:

IOERR – input/output error
EOF – end of volume

EXIT line-ref specifies the line number or label of an EXIT statement to refer to if an error occurs (see "EXIT statement").

Examples

Sample CLOSE statements:

```
20 CLOSE #1:
30 CLOSE #2: IOERR 200
40 CLOSE #3: EXIT CLOSEXT
50 CLOSE #5, FREE: EXIT 400
60 CLOSE #6, RELEASE:
```

Programming considerations

- CLOSE #0 and CLOSE #255
 - CLOSE #0 and CLOSE #255 may be issued even if there was no prior OPEN #0 or OPEN #255.
 - CLOSE #0 and CLOSE #255 may be used to ensure that all screen and printer operations have completed.
- SEQUENTIAL
 - If a DISPLAY I/O file is used for both input and output operations during execution of a single program, the file must be closed and reopened between input and output references.
- Output last data
 - The output file must be closed to make sure that the last records are written. If the diskette is removed without a CLOSE, END, or STOP, data may be lost.
 - Data may be lost if, while running a program, the 5247 Disk or the computer is switched off before a CLOSE, END, or STOP is carried out.

CLOSE

CLOSE statement (continued)

- Extents
 - Any unused portion of the file extents remains assigned to the file after it is closed.

CMDKEY

CMDKEY is a system variable that returns the identity of the last key used to terminate the last INPUT or LINPUT statement.

- -1 is returned if no INPUT or LINPUT was executed in this program.
- 0 is returned if the Enter key was used.
- 1–9 is returned if the Cmd key plus one of the numeric keypad keys was used.

CNT

CNT is a system variable that returns the number of data items successfully processed by the last I/O statement executed. The CNT value is set according to the following:

- CNT is set to 0 before the I/O statement starts executing.
- For INPUT, READ, and PRINT each item is counted as one.
- For LINPUT, the value is set to 1 if data was read.
- For MAT variables, each element is counted separately.

Example:

```
10 OPTION BASE 1
20 DIM A(4)
30 INPUT MAT A
  •
  •
  • (CNT=4 if successful)
  •
```

Note: CNT should be assigned to a variable if its value will be printed. This is because if used in a PRINT statement, CNT will be reset to 0 before the value of CNT is printed. Continuing the above example:

```
40 X = CNT
50 PRINT X
```

CODE

CODE is a system variable that can be set by the program with a STOP or END statement to any value 0 through 9999. It is available to a procedure for testing with the SKIP command.

Commands

See "System commands."

Comments

See "Remarks."

CONTINUE statement

Sets the entire array to a constant (see "ZER and CON").

Concatenation

Concatenation is joining two or more character strings into one. The symbol used for concatenation is the ampersand (&).
For example:

```
10 DIM A$*4
20 DIM B$*3
30 DIM C$*7
40 A$="FLOR"
50 B$="IDA"
60 C$=A$&B$
70 PRINT C$
```

In this example, the character string A\$ is concatenated (&) with the character string B\$ to form string C\$ (FLORIDA).

Another example of how to use concatenation is as follows:

```
10 LET A$="MIKE"
20 LET B$="//1"
30 OPEN #1:"NAME="&A$&B$&"," ,SIZE=0,
      RECL=127",INTERNAL,OUTPUT
40 CLOSE #1:
```

Line 30, above (using concatenation) is the same as line 10, in the following:

```
10 OPEN #1:"NAME=MIKE//1,SIZE=0,
      RECL=127",INTERNAL,OUTPUT
20 CLOSE #1:
```

The result of concatenation must be 255 characters or less.
For more information, see "Character expression."

CONTINUE statement

The CONTINUE statement transfers control to the statement following the one causing the most recent ON-condition transfer or I/O exit.

CONTINUE _____■

CONTINUE is useful following an ON GOTO transfer or I/O exit. If an ON event is specified to be ignored, the return statement specification used by CONTINUE is not changed. See "ON statement."

If a second ON GOTO or I/O exit occurs before CONTINUE is executed, the first occurrence is lost. Avoid operations that cause such occurrences or use ON. . . IGNORE.

If no error has occurred since RUN, execution of CONTINUE causes an error and interrupts execution of the program.

Any event that causes an ON GOTO transfer or I/O exit including the Inq key (ON ATTN), sets the CONTINUE target line.

For a description of special handling of ON events and I/O exits within a defined function, see "DEF, FNEND statement."

COS(X)

COS(X)

Returns the cosine of X, where X is in radians. The absolute value of X must be less than 1E10. For best accuracy specify a value for X greater than -2π or less than 2π .

Note: For correct results, use: $\text{COS}(X)=2*(\text{COS}(X/2)**2)-1$. (This note does not apply if the system release code field on your system status line is "1.05.")

Cross reference

See "LIST label" under "LIST,LISTP command."

Customer Support Functions

The Customer Support Functions are supplied by IBM on diskettes. For detailed information about Customer Support Functions, see *Customer Support Functions*, Volume I and Volume II. The Customer Support Functions are:

- Select Machine Update
- Load Machine Update
- Prepare Diskette
- Copy
- Display Diskette Label
- Recover Diskette
- Create Index File
- Change Collating Sequence
- Replace Customer Support Function
- Prepare Sort Control File
- Sort
- List Diskette or Disk Information
- List Files
- List Storage Information

The following Customer Support Functions are part of the Communications Licensed Programs:

- Setup Asynchronous Communications
- Setup Binary Synchronous Communications
- Prepare Batch Data Transfer
- Batch Data Transfer
- Asynchronous Communications Terminal
- Asynchronous Problem Information
- Binary Synchronous Problem Information

Customer Support Functions

Customer Support Functions (continued)

- BSC Online Test

The following Customer Support Functions are for the 5247 Disk:

- Add or Delete a Volume
- Backup and Restore
- Prepare Disk
- Recover Disk

Note: Some Customer Support Functions may be called and controlled from Procedure Files

Select Machine Update—LINK SELECT

The Select Machine Update function creates a file of machine updates to be used by the Load Machine Update function. This must be done before the Load Machine Update features can be performed.

Load Machine Update—LINK UPDATE

The Load Machine Update function loads machine updates (supplied by IBM) into the system.

Prepare Diskette—LINK PREPARE

The Prepare Diskette function prepares a new diskette or erases a used one. A new diskette cannot be used as it is received, it must be prepared to the format required by the system.

Customer Support Functions

Customer Support Functions (continued)

Copy—LINK COPY

The Copy function does any of the following:

- Copies an exact image of an input diskette to an output diskette
- Copies a group of files from an input disk volume or diskette to an output disk volume or diskette
- Copies a selected input file to an output file
- Copies a selected input file or group of files to the printer
- Copies all files from an input diskette to an output diskette
- Compresses files by eliminating deleted records or unused extents

Display Diskette Labels—LINK LABEL

The Display Diskette Labels function displays the contents of the diskette labels for use in recovery procedures. The contents of these labels can also be printed. Labels on any access-protected diskettes cannot be displayed or printed. See "DIR command" and "VOLID command."

Recover Diskette—LINK RECOVER

The Recover Diskette function recovers a file when a read error occurs on the label or data portion of the file. The function will save as much of the data on the file as possible. Accidentally freed or dropped files can be recovered even though no read error occurred.

Create Index File—LINK INDEX

The Create Index File function creates index files for use in accessing master data file records (see "Key-indexed files").

Change Collating Sequence—LINK COLSEQ

The Change Collating Sequence function replaces the memory-resident collating sequence with an alternate collating sequence. It also modifies the active collating sequence from the keyboard.

For related BASIC subjects see:

"IF statement"

"AIDX and DIDX"

"OPTION statement"

Replace—LINK REPLACE

The Replace function finds obsolete versions of the Customer Support Functions and replaces them with newer versions.

Customer Support Functions

Customer Support Functions (continued)

Prepare Sort—LINK PRESORT

The Prepare Sort function defines the files, sorts fields, and other information to be used by the Sort function. Prepare Sort must be performed before the Sort function can be used.

Sort—sort-control-file

The Sort function performs a *Record-Out Sort* or an *Address-Out Sort*. Record-out sort creates a new file with the *records* sorted. Address-out sort creates a new file with the address (positions) of the record in the sorted order.

List Diskette or Disk Information—LINK LISTDISK

The List Diskette function gives you information about the files on a disk volume or diskette.

List Files—LINK LISTFILE

The List File function lets you investigate the records in a file. It uses your answer to prompts to list records and summary information about the records.

List Storage Information—LINK LISTSTOR

The List Storage function is a tool that helps you debug a BASIC program. It will print and display various parts of storage that were previously saved on diskette by the built-in diagnostic dump. The storage is interpreted in terms of the BASIC program which was resident at the time of the dump.

Setup Asynchronous Communications—LOAD SETUP.ASC

The Setup Asynchronous Communications function creates a file containing the communications environment data. This function must be done before Communications can be run.

Setup Binary Synchronous Communications—LOAD SETUP.BSC

The Setup Binary Synchronous Communications function creates a file containing the binary synchronous communications environment data. This function must be done before Communications can be run.

Prepare Batch Data Transfer—LOAD PREBDT

The Prepare Batch Data Transfer communications function builds a control file that directs the operation of Batch Data Transfer.

Batch Data Transfer—LOAD BDT

The Batch Data Transfer communications function transfers data files to and from a remote system.

Customer Support Functions

Customer Support Functions (continued)

Asynchronous Communications Terminal—LOAD ACT

The Asynchronous Communications Terminal function operates as an interactive terminal for asynchronous communications.

Asynchronous Problem Information—LOAD PROBLEM.ASC

The Asynchronous Problem Information communications function displays trace and statistical information from an asynchronous communications session.

Binary Synchronous Problem Information—LOAD

PROBLEM.BSC

The Binary Synchronous Information communications function displays trace and statistical information from a binary synchronous communications session.

BSC Online Test—LOAD OLTST

The BSC Online Test function performs Binary Synchronous Communications online tests to verify the communications link.

5247 Disk Customer Support Functions

Add or Delete a Volume—LINK ADD.DELETE

The Add or Delete a Volume function creates a new volume on the disk or deletes a volume from the disk.

Backup and Restore—LINK BACKUP.RESTORE

The Backup and Restore function copies disk volumes to diskettes so that you can create backup files. These backup files cannot be accessed by BASIC commands until they are restored to the disk.

Prepare Disk—LINK PREPARE.DISK

The Prepare Disk Function prepares the 5247 Disk to the format required by the system.

Recover Disk—LINK DISK.RECOVER

The Recover Disk Function recovers accidentally dropped or freed files on the disk. Options are included in this function to allow you recover from read errors on a file or on data set labels.

Data buffering

Data buffering

The term *buffering* means storing data in an intermediate storage area when coming or going to an I/O device.

The System/23 reserves sufficient storage to perform any valid I/O operation, once the file has been opened. Substantial improvements in performance can be realized by allowing (or adding) additional storage which can be committed to the OPEN operation. This allocation is handled automatically by the System/23 whenever storage is available. If the size of your program does not permit this allocation, it will still function correctly, but slower.

The maximum space used in this allocation is 512 bytes for each file that is an index file or is being opened for relative access.

A larger storage allocation is made for files that are open as:

- DISPLAY,INPUT
- DISPLAY,OUTPUT
- INTERNAL,INPUT,SEQUENTIAL
- INTERNAL,OUTPUT,SEQUENTIAL
- PROC
- SUBPROC

For the above type of open files the space used is the minimum of the following:

- 7680 bytes for type 2D diskettes and disk files
- 4096 bytes for type 1 and 2 diskettes
- The extent size to be read
- The value of the SIZE= parameter on the OPEN for output of the file

DATA files

Type 05 (DISPLAY) files are used in the System/23 for procedures, as input to programs (INPUT and LINPUT), and for any other data in the form of keyed input or printed output. With System/23, you can create, view and edit DISPLAY files. To create a DISPLAY file, enter:

```
CLEAR DATA  
AUTO
```

CLEAR deletes any previous data or program from the work area. AUTO puts a line number and colon on the input line. Now enter any desired data, such as procedure file commands or data. All normal editing facilities are available.

When finished, enter:

```
SAVE file-spec
```

This puts the new file on the diskette or disk. Remember to include either a VOLID or device address in the file specification. To edit an existing DISPLAY file, enter:

```
LOAD file-spec, DATA
```

Next, edit the file as usual, then enter:

```
REPLACE
```

For additional information see, "CLEAR command" and "Editing a program."

DATA files

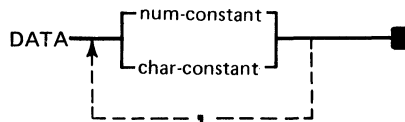
DATA files (continued)

The following BASIC program prints the DATA file (type 05) on the system printer.

```
10 DIM LINE$*255
20 OPEN #1: "NAME=file-spec", DISPLAY, INPUT
30 LOOP: LINPUT #1:LINE$ EOF QUIT
40 PRINT #255: LINE$
50 GOTO LOOP
60 QUIT: STOP
```

DATA statement

The DATA statement creates an internal data table. The data table constants are assigned to the variables and/or array elements by the READ statement (see "READ statement").



The syntax of the DATA statement is as shown above, where:

num-constant is any numeric value (see "Arithmetic constant" under "Arithmetic data").

char-constant is any character string value. The character string may be quoted or unquoted. In the quoted character string, any characters are allowed. In the unquoted character string, leading and trailing blanks are ignored, commas and quotes are not allowed.

When program execution begins, a pointer is set to the first constant in the table. The pointer advances as data is read by

the READ statement. (The RESTORE statement may be used to restore the pointer to the first constant.)

Example

```
100 OPTION BASE 1
110 DATA "DEBIT",21.60,"CREDIT",15.40
120 DATA MONTH, DAY, YEAR
130 READ A$,N,B$,C
140 DIM Z$(3)
150 READ MAT Z$
```

Programming considerations

- Too few values
 - If the DATA statement does not contain enough constants for the READ statement issued, an EOF error is generated.
- Character data
 - Character data does not have to be enclosed in quotation marks unless leading blanks, embedded commas, or lowercase characters are significant. Unquoted lowercase letters and graphic characters are converted to uppercase.
- Numeric data
 - Numeric values may be accessed and read as either a numeric or character value.
- Remarks
 - A remark is not permitted on DATA statements. It is interpreted as part of the data.

DATE

DATE command

The DATE command assigns the specified date to the system variable DATE\$.

DATE—yy/mm/dd —■

The syntax of the DATE command is shown above, where:

yy is in the range 00 to 99 *mm* is in the range 01 to 12 *dd* is in the range 01 to 31

An example of the DATE command with a remark is:

```
DATE 81/01/01 ! Happy New Year
```

DATE\$

DATE\$ returns an eight character string that is set by the DATE command. When power to the computer is switched on, DATE is set to (b b / b b / b b).

The date is *not* updated by the system.

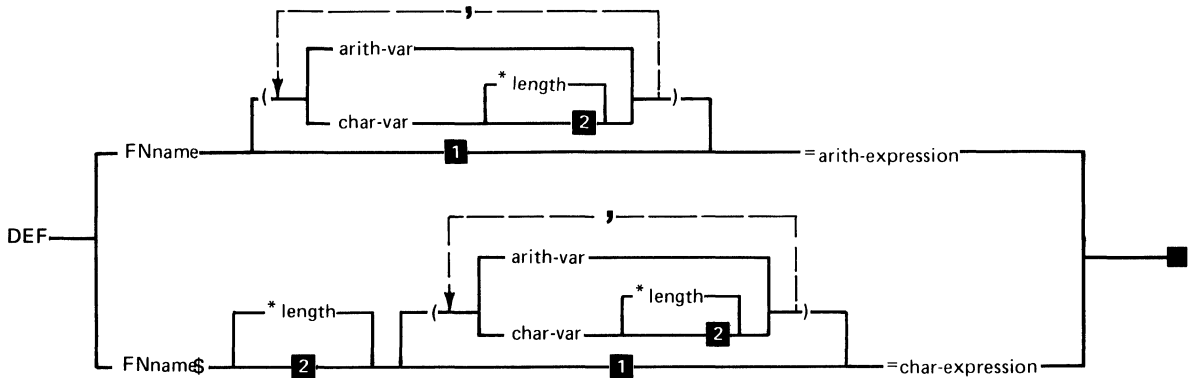
Declaring arrays

See "DIM statement."

DEF, FNEND statement

The DEF statement is used to define an arithmetic or character valued function for reference elsewhere in the program. The FNEND statement indicates the end of a multiple-line function. The syntax of the DEF statement can be either a one-line or multiple-line function.

One-line function



- 1** No input parameters
- 2** Length is 18

The syntax for the one-line function is shown above, where:

FNname is any valid variable name. This name, preceded by the letters FN is the name of the defined function. For character valued functions, this name must be followed by the dollar sign \$.

DEF, FNEND

DEF, FNEND statement (continued)

length is the length of the character variable used as input or output. The length may range from 1 to 255 characters.

arith-var is an arithmetic variable name to which a value is assigned when the function is called.

char-var is a character variable name to which a value is assigned when the function is called. Values assigned to the character variable cannot exceed the maximum length of variable, or loss of data will result.

arith-expression is an arithmetic expression that specifies the value to be returned for the function. If the function name is an arithmetic variable, an arithmetic expression must be specified. See "Arithmetic expressions."

char-expression is a character expression that specifies the value to be returned for the function. If the function name is a character variable, a character expression must be specified. See "Character expressions."

Example one-line DEF statements

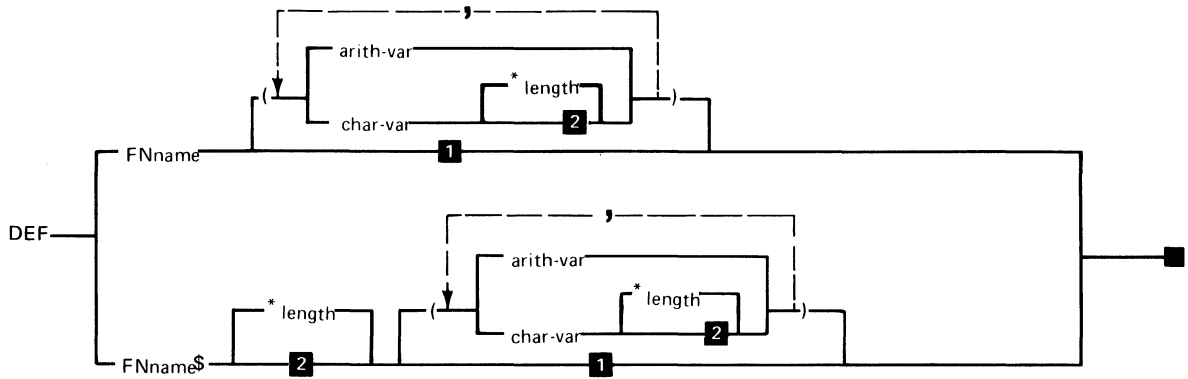
Arithmetic function:

```
120 DEF FNA(R)=2*R+100
```

Character function:

```
120 DEF FNA$(R)=STR$(R+5)
```

Multiple-line function



- 1** No input parameters
- 2** Length is 18

The syntax for the multiple-line function is shown, where:

FNname is any valid variable name. This name, preceded by the letters FN, is the name of the defined function. For character valued functions, this name must be followed by the dollar sign.

DEF, FNEND

DEF, FNEND statement (continued)

length is the length of the character variable used as input or output. The length may range from 1 to 255 characters.

arith-var is an arithmetic variable to which a value is assigned when the function is called.

char-var is a character variable to which a value is assigned when the function is called. Assigned values for the character variables cannot exceed the maximum length of the variable.

The LET statement assigns the value of an expression as the result of the function.

The FNEND statement is descriptive and indicates the end of a multiple-line function. The value of the function is specified in an expression in the LET statement.

Example multiple-line DEF statement

```
10 LET A=5
20 LET B=2
30 LET C=-5
40 DEF FNA(X,Y)
50 IF X > 0 THEN LET FNA=X+Y ELSE LET FNA=X-Y
60 FNEND
70 LET D = FNA(A,B)
80 LET E = FNA(C,B)
90 PRINT D,E
```

When the statements in this example are executed, D will have a value of 7 and E will have a value of -7.

DEF, FNEND

DEF, FNEND statement (continued)

The use of functions

When a user defined function reference appears in an executable BASIC statement, any expressions that follow the function name must be separated by commas and enclosed in parentheses. These expressions are evaluated and passed by the system to the user function in order to initialize the corresponding variables in the DEF statement. These values must agree in number, length, and type with the corresponding variables in the DEF statement. If the DEF expression is present, the function is defined on the same line and its value is the value of that expression. This is a one-line function. If no expression is specified in the DEF statement, the DEF statement is the start of a multiple-line function. In this case, the FNEND statement indicates the end of the function and the value of the function is specified by the value of the variable FNname assigned in the LET statement.

Programming considerations

- Use of functions
 - A function reference to a user-defined function may appear anywhere in a BASIC program that a constant, variable, subscripted array element reference, or system function reference can appear (see “Arithmetic expressions”).
- Location
 - A function can be defined anywhere in a BASIC program, either before or after it is referenced.
- Name localization
 - The variables named in the DEF statement are local to the function. Consequently, it is possible to have a variable in the DEF statement with the same name as a variable used elsewhere in the program. Each variable is recognized as being unique, and no conflict of names or values results from this duplicate usage. All variables which are not DEF arguments have the same value and meaning for all statements.
- Assigned value
 - The value of a defined function is assigned when the function name appears to the left of a LET statement and the corresponding defined function is referenced in the program. The LET statement usually appears within the defined function. The value may also be defined as a result of another defined function reference, a GOSUB, GOTO, ON-condition transfer or an I/O error exit. For example:

DEF, FNEND

DEF, FNEND statement (continued)

```
10 LET A=FNA+R
20 DEF FNA
30 F=10
40 IF F>6 THEN R=FNB ELSE FNA=30
50 FNEND
60 DEF FNB
70 FNB=60
80 FNA=90
90 FNEND
100 Print A
```

- The result of this program is: A=150.
- On **CONDITION** localization
 - When execution of a multiple-line defined function begins, all **ON CONDITION** settings are stacked and set to **SYSTEM**. New settings for the **ON CONDITION** may be specified within the function. If an **ON** event occurs within the function and the specification is **IGNORE**, it will be ignored. If the specification is **GOTO**, the transfer will occur, and the function will remain active. **CONTINUE** and **RETRY** will return execution to the appropriate line within the function. If **SYSTEM** is active, the function execution is abandoned. The **ON** conditions are unstacked and whatever was specified for the event preceding the current function will occur (**IGNORE**, **SYSTEM**, **GOTO**).
- **Bypass function**
 - After control is passed to a **DEF** statement without reference to the function, control goes to the first executable statement following the function definition (the **DEF** statement for one-line functions, or the **FNEND** statement for multiple-line functions).
- **Cmd/Attn**
 - If **Cmd/Attn** is pressed during the execution of a defined function, execution is interrupted. If **Error**

Reset is pressed, execution of the function will resume normally. If Cmd/Error Reset is pressed the function is abandoned and the system enters split screen mode at the line which invoked the defined function.

- I/O exits
 - Exit clauses specified in I/O statements within a multiple-line defined function causes the specified transfer of control when the event occurs. The function remains active and CONTINUE and RETRY will return to the appropriate line within the function.
- Inq
 - If the Inq key is pressed during execution of a defined function, it is ignored until all executing, defined functions are finished. At that time, the ON action specified prior to entering the function, occurs. (IGNORE, SYSTEM, GOTO).
- Single definition
 - A function of a given name can be defined only once in a given program.
- Recursion
 - A function cannot contain references to itself or to other functions that refer to it.

DEF, FNEND statement (continued)

- FOR/NEXT
 - A FOR/NEXT loop beginning in a function must also end in the same function.
- Nesting
 - DEF function definitions cannot be nested.
- Input/Output
 - User-defined functions that are referred to during an input or output operation cannot themselves perform any input or output operation.
- Modification of variables
 - If a function definition alters the value of a variable that is referenced in the same statement that calls the function, the results may not be as expected.
- Termination
 - A program may not be terminated when a defined function is still in execution. An FNEND must be issued for each invoked DEF before the program ends.
- EXIT and FORM
 - EXIT and FORM statements inside a multiple-line DEF function can be referenced from outside the function; those outside the function can be referenced from inside a DEF function.
- Be sure that the first line of a multiple-line function (DEF) is not the last line of the program.

DEL command

The DEL command deletes one or more consecutive lines from a BASIC program or DATA work area.

DEL— first line-num — **1** — last line-num

1 Delete only one line number

The syntax of the DEL command is shown above, where:

first-line num is a number specifying the first line number of several consecutive line numbers to be deleted. It may also be the only line number to be deleted.

last-line num is a number specifying the last line number of several line numbers to be deleted.

The numbers used in the first-line and last-line numbers must be integers in the range of 1 through 99999. The first-line number must be less than the last-line number.

No additional storage becomes available as a result of using the DEL command. The additional storage will become available when the program is saved in SOURCE format and loaded.

DEL

DEL command (continued)

Example

To delete line 20 from a program or data work area:

```
DEL 20
```

To delete lines 20 through 90 from a program or data work area:

```
DEL 20,90
```

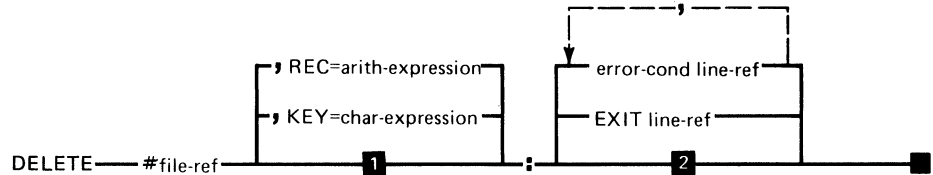
If line 20 or 90 does not exist in the workspace, then the range of lines that do exist between 20 and 90 will be deleted. If no line exists, an error is presented.

Programming considerations

- Comments should not be used after the DEL command.

DELETE statement

The DELETE statement deletes either the last record read from the file or the record specified by the position specification. After the record is deleted, the file is positioned to a location immediately following the deleted record.



- 1 DELETE last record accessed READ/REREAD
- 2 Interrupt on error unless ON is active

The syntax of the DELETE statement is shown, where:

file-ref is a numeric expression (see “File reference parameter”).

REC=arith-expression specifies that the record having a record number equal to the arithmetic expression is to be deleted.

KEY=char-expression specifies that the first record in the file having a key equal to the character expression is to be deleted.

error-cond line-ref is the error action for one of the following: NOREC, IOERR, or NOKEY

For information on the error actions, see “EXIT statement.”

DELETE

DELETE statement (continued)

EXIT line-ref specifies the line number or label of an EXIT statement to refer to if an error occurs.

Example

A sample DELETE statement is shown here:

```
80 A$="ZEPOL"  
90 DELETE #8, KEY = A$:
```

In this example, the first record with a key field equal to ZEPOL is deleted.

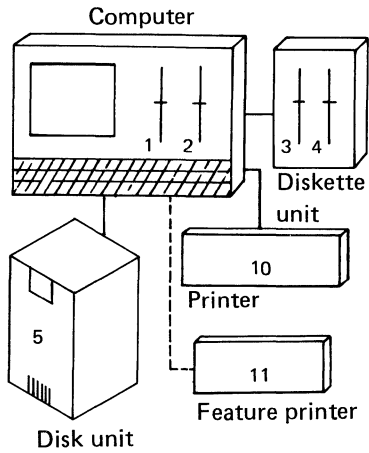
Programming considerations

- The file must have been opened as INTERNAL, OUTIN. The file organization may be SEQUENTIAL, RELATIVE, or KEYED.
- The SEARCH parameter is not permitted.
- If no KEY or REC parameter is specified, the previous access to this file must have been a successful READ or REREAD statement.

Descending index (MAT assignment)

See "AIDX and DIDX."

Device address parameter



Many BASIC statements and system commands require entry of a device address parameter. This address identifies the input/output device being used. Valid device addresses (in decimal) for System/23 are:

- 1 Diskette drive 1
- 2 Diskette drive 2
- 3 Diskette drive 3
- 4 Diskette drive 4
- 5 5247 Disk unit
- 10 Printer
- 11 Feature printer
- 40 Communications

See "OPEN statement," and "File specification parameter."

Sharing the 5246 Diskette Unit

Device sharing with the 5246 Diskette Unit means that two 5322 Computers are connected to the same 5246 Diskette Unit and both have different open files on the 5246. This situation is handled entirely by the system and never produces any new logical or data integrity questions.

The 5246 can only service one computer at a time and thus each computer may experience additional waiting time when the other computer is already using the 5246. For this reason, only data that is to be shared by both computers should be located on the 5246. Files that are to be used only by one of the computers should be located on that computer's built-in drive.

After you switch on power, an attempt is made to access the 5246 to establish its presence and the computer work station identification (WSID\$).

- If the 5246 is in use by the other computer, +0 appears on the status line. In this case, the operator may simply wait for the 5246. If the operator does not wish to wait, then he may press Cmd/Attn, producing an action code 21 and error code 6009. ERROR RESET will now return the computer to waiting for the 5246.

Cmd/Error Reset terminates the wait. The 5246 becomes logically detached from the computer and all future references to it will cause an error 4153 (device not attached). To attach the 5246, you must now switch the computer off and on again, with the 5246 power switched on.

- If the 5246 is not switched on, action code 21, error code 6009 is displayed. To attach the 5246, switch it on and press Error Reset. If the 5246 is not plugged in or has a blown fuse, action 21, error 6009 will reappear. To

continue to work without the 5246, press Cmd/Error Reset.

The state of the shared 5246 is shown on the status line in columns 53 and 54 by the following codes:

- blank—The 5246 is not required by this computer.
- +0—The other computer is currently using the 5246, or the 5246 was switched on before the computer, but is now switched off; this computer is waiting to use it. If the 5246 is switched on and you wish to wait for it, ---DO NOTHING---
- +1—This computer is now using the 5246.

When the computer is waiting to use the 5246 (+0), the operator may interrupt this wait with Cmd/Attn. To continue with the original operation, press Error Reset. To end the current operation with I/O error 6009 or 6011, press Cmd/Error Reset. The latter action ends a program (with GO END) or command if the other computer will be using the 5246 for a long time.

If the program has open files when you do this, data may be lost.

For some operations, multiple I/O accesses may be attempted following the interrupt of the +0 wait. After the first Cmd/Attn, Cmd/Error Reset cycle, another +0 may appear. Continue with the Cmd/Attn, Cmd/Error Reset cycle until the +0 is cleared.

Share and Reserve status may be left on for the file which was being accessed. Use the PROTECT command to remove them.

Pressing Hold while +1 is displayed, stops the 5246 and prevents the other computer from using it.

Device sharing

Device sharing (continued)

The 5246 is also unavailable for the duration of an action code 10 (waiting for diskette to be inserted), if drive 3 or 4 or no drive is given in the file specification. This will also occur if the diskette contains an open file, the diskette has been removed, and is now required.

Each of the two cables connecting a computer to the 5246 identifies the connected computer with respect to the use of the 5246. This identification is provided through the `WSID$` system variable.

If the computer is connected to cable 1, or is not attached to the 5246, or the 5246 power is off when the computer is switched on, then `WSID$="01"`

If the computer is connected to cable 2 and the 5246 was switched on before the computer, then `WSID$="02"` `WSID$` is useful in establishing unique file names when the same application is running in both computers.

The following Customer Support Functions secure the 5246 Diskette Unit while they are running:

- Prepare Diskette
- Copy (image copy only)
- Recover Diskette
- Display Diskette Label

See also "File sharing" for information on the simultaneous use of the same file by two different opens.

Sharing the 5247 Disk Unit

The 5247 Disk Unit lets up to four computers share information. Sometimes you may have to wait, if the information you wish to use is in use by another computer. The length of the wait depends upon what the other computer is doing to the information.

To find out if another computer is using the disk, look on the status line in columns 56 and 57. These columns are blank when you are not using the disk. Whenever you request to use the disk, the computer reports the sharing status by displaying either a *1, *2, *3, or *4 in columns 56 and 57.

***1** Your computer is requesting to use the disk. No action is required of you.

***2** Another computer has exclusive use of the disk. You will not be able to use it until the other computer has completed its operation.

If you cannot wait for the disk to become available, press the CMD/Attn key and then press the CMD key and the Error Reset key. This clears the *2 from the status line. Note, however, that this is not recommended as a normal procedure as data may be lost. Error Reset returns the system to waiting.

***3** The disk is processing a request from a computer. No action is required of you.

***4** Another computer is using the file that you are requesting to use. In this case, the other computer will not allow you to share the file.

If you cannot wait for the locked file to become available, press the Cmd/Attn key and then press the Cmd key and the Error Reset key. This clears the *4 from the status line. Note, however, that this is not recommended as a normal procedure,

Device sharing

Device sharing (continued)

as data may be lost. Error Reset returns the system to waiting.

See also "File sharing," "RESERVE statement" and "OPEN statement" for information on the simultaneous use of the same file.

DIDX (array name)

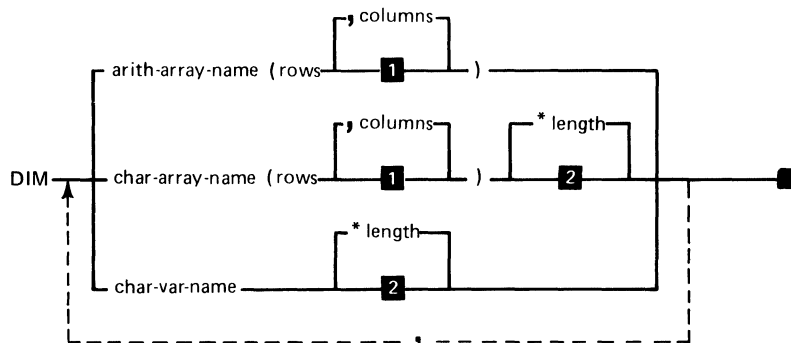
DIDX returns an array containing the descending index of the source array. See "AIDX and DIDX" Also see "OPTION statement" (COLLATE).

Dimensioning arrays

See "DIM statement."

DIM statement

The DIM (dimension) statement specifies the maximum size of arrays and character variables, and their original dimension.



1 One-dimensional array (vector)

2 Defaults to length of 18

The syntax of the DIM statement is shown above, where:

arith-array-name is an arithmetic array to be dimensioned.

char-array-name is a character array to be dimensioned.

char-var-name is a character variable to which a length will be assigned.

rows and columns are integers specifying the dimensions of the arrays (highest subscript(s)). One dimensional arrays require only the row entry. Two dimensional arrays require both the row and column entries separated by a comma. For example:

```
10 DIM A(20,25)
```

DIM

DIM statement (continued)

length is the maximum length of a character scalar, or the maximum length of each element of a character array. This value may be from 1 to 255. If length is not specified, the default maximum length is 18 characters.

The initial value of each arithmetic array element is zero. Each character array element is initialized to null (zero length). This initialization takes place when the array is first referenced.

Declaring arrays

Arrays can be declared either by using the DIM statement or by a reference to an element of an array that has not been declared.

When an array is declared by using the DIM statement, the dimension and maximum number of data items are specified in the DIM statement. For example:

```
20 DIM A(10)
30 DIM WEEK$(6)*9
```

Statement 30 dimensions the array named WEEK\$ to use the seven array elements WEEK\$(0) through WEEK\$(6). The maximum length of each element is nine characters.

If OPTION BASE 0 is in effect, statement 20 dimensions the array named A to use 11 array elements A(0) to A(10).

When an array is declared by a reference to one of its elements, it is one- or two-dimensional, based upon its use. It has 10 elements in OPTION BASE 1 and 11 elements in OPTION BASE 0. For more information, see "OPTION statement." For example:

```
40 A(3) = 50
```

establishes a one-dimensional array containing 10 elements, if `OPTION BASE 1` is in effect, the third element `A(3)` has an integer value of 50, and the remaining elements have values of zero. For example:

```
50 WEEK$(0) = "Sunday"  
60 WEEK$(3) = "Wednesday"
```

Arrays requiring more than 10 elements (`BASE 1`) or 11 elements (`BASE 0`) must be declared explicitly.

An array can be declared by a `DIM` statement only once in a program.

If an array or character variable is passed as a parameter by the `CHAIN` statement, it must be declared in a `DIM` statement in both programs, and the same size must be specified.

For more information see:

- "Character arrays"
- "Arithmetic arrays"
- "Substring referencing"

DIM

DIM statement (continued)

Example

A sample DIM statement is shown:

```
10 OPTION BASE 1
20 DIM A$(5)*20,B(4,2)
30 LET X = LEN(P$) ! x is 0
40 LET P$= "ABCDEFGHJKLMNOPQR"
```

The result of the DIM statement is:

A\$ is a character array with five elements (one-dimensional array); each element has a maximum of 20 characters. All five elements are initialized with a length of zero.

B is an array of four rows and two columns (two-dimensional array).

P\$, which is not declared in any DIM statements defaults to a maximum of 18 characters and is initialized with a length of zero. Statement 40 changes its length to 18.

Programming considerations

- Redimensioning
 - If you want to change the size of an array during execution time, redimensioning can be used. However, the array is allocated to its full dimensioned size when first referenced. The storage will be reused when redimensioning occurs. Another technique is to create a procedure file to edit a DIM statement into the program. For example:

```
LOAD file-spec
10 DIM A$(59)*33
RUN
•
•
•
```

See "Redimensioning."

- *Zero
 - If a length of 0 is specified, it is interpreted as the default length of 18.
- Duplicate DIM
 - An array or character variable cannot appear in a DIM statement if it has already been defined in another DIM statement.
- The maximum value that can be specified for row or column is 9999. If sufficient storage in the work area is available, the maximum size of an array is 65534 (see "Storage use").

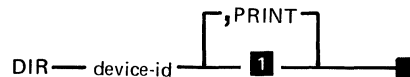
DIR

DIR command

The DIR command lists a directory of file information. One line of information for each file is printed or displayed by the computer.

Using the DIR command when you do not have a 5247 Disk Unit

DIR— device-id — [1] — [PRINT] —



1 Displays the directory on the screen

The syntax of the DIR command, when you do not have a 5247 Disk Unit is shown above, where:

device-id specifies which diskette drive is to have its files listed. The devices are 1, 2, 3, and 4 (see "Device address parameter").

- Diskette drive 1
- Diskette drive 2
- Diskette drive 3
- Diskette drive 4

PRINT specifies that the listing be printed (device address 10).

The listing can be interrupted by pressing the Hold key once. To continue with the listing press the Hold key once again. To terminate the commands press the Cmd/Attn key. Since printer operations overlap other System/23 operations, after pressing the Cmd/Attn key, the printer will print the data remaining in the print buffer.

DIR command (continued)

Example

The following information is displayed about each diskette.

- 1** The VOLID (volume identification) of the diskette
- 2** The diskette type (1, 2, 2D)
- 3** The number of bytes not used by files on diskette
- 4** The number of available files
- 5** The number of defective sectors
- 6** The physical record size

Additionally for each file on the diskette, DIR will display:

- 7** File type (see "Diskette file types")
- 8** filename (see "File specification parameter")
- 9** Number of bytes allocated to the file
- 10** Number of bytes of data in the file
- 11** The number of extents in the file
- 12** Protective information; P means protected, Read only allowed

The following are File Sharing Status:

- 13** Station 1 Open status
- 14** Station 1 Reserve status
- 15** Station 2 Open status
- 16** Station 2 Reserve status

DIR 1

1	2	3	4	5	6					
CONKLN	2D	0910336	0046	0000	512					
05	AUTO		0001024	0000512	0001					
05	FAIRWAY		0003072	0003072	0005					
05	FSP.SOURCE		0001536	0001024	0001					
09	ANIMATE		0008192	0007168	0001					
05	PROCL		0000512	0000512	0001					
05	DEMO		0000512	0000512	0001		ISI			
07	FILE.IND		0000512	0000512	0001					
05	CH		0002048	0001024	0001					
05	MAKE.SCREEN.SRCE		0002048	0001024	0001					
04	NEWFILE		0004096	0000972	0001	P				
05	SCREEN1		0004096	0002048	0001					
05	NEWINDEX		0000512	0000512	0001					
05	FSP.TEST		0004096	0001024	0001					
05	BUILD.MURPHY		0001024	0000512	0001					
05	MURPHY.FIX		0004096	0001024	0001			OSH	OSH	
05	MURPHY 0003584		0002048	0001024	0001		ISH			
04	MURPHY.FILE		0025088	0024920	0001					
04	SCREEN.FILE		0020480	0020480	0001		NS			
05	SCREEN		0002560	0001536	0001					
7	8		9	10	11	12	13	14	15	16

DIR

DIR command (continued)

The five types of file-sharing are:

NS – opened no share

ISH – opened for input, SHR

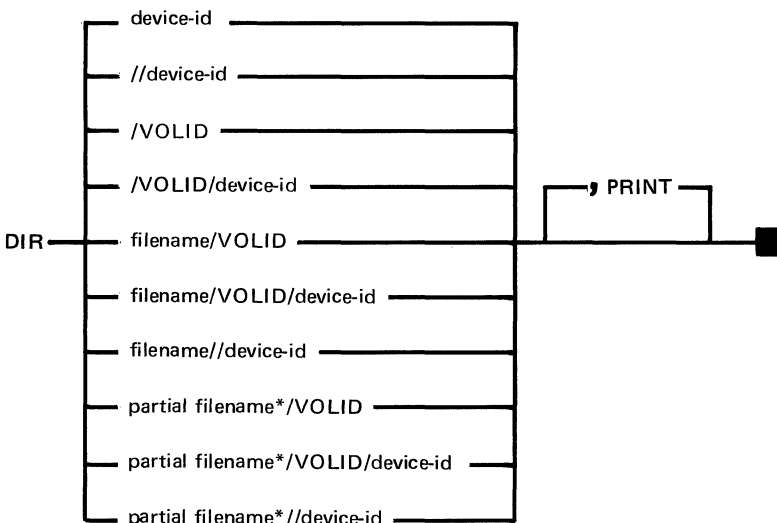
ISI – opened for input, SHRI

OSH – opened for output, SHR

OSI – opened for output, SHRI

The preceding information is displayed for System/23 type diskettes (type Z) and BX and HX diskettes. For diskettes containing BX or HX files, the volume information (first line) will contain only VOLID, diskette type, and physical record size. File information will not contain sharing or reserve status. If other diskettes conforming to IBM diskette data format are used, only the VOLID and FILEID are accurate. The rest of the information will be unpredictable.

Using the DIR command when you have a 5247 Disk Unit



The syntax of the DIR command, when you have a 5247 Disk, is shown above, where:

device-id or *//device-id* lists all the files in the specified drive number. The devices are 1,2,3,4 (diskette drives) and 5 (disk unit). If drive 5 is specified, the list will show all the files on the first volume only of the disk. See "Device address parameter"

DIR

DIR command (continued)

/VOLID lists all the files in the specified volume. See "VOLID command" and "File searches."

/VOLID/device-id lists all the files in the specified volume of the specified drive number.

filename/VOLID gives a line of information for the named file on the specified volume.

filename/VOLID/device-id gives a line of information for the named file on the specified volume of the specified drive number.

filename//device-id gives a line of information for the named file in the specified drive number. If drive 5 (disk unit) is specified, the information for the file is from the first volume only on the disk.

partial filename/VOLID* lists all the files that begin with the same letters specified, on the specified volume. For example, DIR SET*/VOL002 lists all the files on VOL002 that begin with the letters SET.

partial filename/VOLID/device-id* lists all the files that begin with the same letters specified, on the specified volume of the specified drive number.

partial filename//device-id* lists all the files that begin with the same letters specified, in the specified drive number. If drive 5 (disk unit) is specified, the list of files is from the first volume only on the disk.

PRINT specifies that you want the listing to be printed.

Example

DIR device-id

1	2	3	4	5	6	13	14	15	16	17
VOL001	30180864		0091	0000	512					
10	COPY		00030208		00030208	0001	ISI			
04	FILE.ONE		00024576		00016200	0045	ISH	OSI	OSI	
05	FILE.TWO		00024576		00015872	0045		OSH		2
09	PROGRAM.FILE		00002560		00002560	0001	P			
04	FILE3		00002048		00000000	0001	OSU			
05	FILE4		00005120		00000512	0001	ISU			4
04	FILE.INTERNAL		00003584		00000957	0001				1*
04	INTERNAL.FILE		00002048		00000000	0001	NS			
7	8	9	10	11	12	18				

The screen displays the following information about each diskette (if drive 1,2,3, or 4 is specified) or for the first volume on the disk (if you specify drive 5) or for the volume specified if the VOLID parameter is used.

- 1 The VOLID of the disk volume or diskette.
- 2 The diskette type (type 1,2,or 2D). This field is blank when the volume is on the disk.
- 3 The number of available bytes on the disk volume or diskette.
- 4 The number of available files on the diskette. For disk volumes, this is the number of file labels available in the current allocation. When all file labels in the current allocation are used, an attempt is made to allocate 100 more.

DIR command (continued)

If space is available on the disk, this will be successful until a volume contains 9998 files.

- 5** The number of defective sectors on the diskette. This number does not apply to volumes on the disk.
- 6** The physical block size.

For each file on the disk volume or diskette, DIR displays:

- 7** The file type (see "File types").
- 8** The filename (see "File specification parameter").
- 9** The number of bytes allocated to the file.
- 10** The number of bytes of data in the file.
- 11** The number of extents in the file.
- 12** File protection information: a P in this field means that the file is protected and READ only is allowed.

The following fields show the file sharing information for each file:

- 13** The file-sharing information for computer #1. If the volume is on the disk, file-sharing information for the computer you are using.
- 14** The RESERVE sharing information for computer #1. If the volume is on the disk, the RESERVE sharing information for the computer you are using.
- 15** The file-sharing information for computer #2. If the volume is on the disk, the most restrictive sharing information for the other computers using the file. If the field is blank, no other computers are using the file.
- 16** The RESERVE sharing information for computer #2. If the volume is on the disk, the most restrictive RESERVE information for the other computers reserving the file. If the field is blank, no other computers are using the file.

- 17** Which computer has a RESERVE statement in effect for the file. If blank, no RESERVE is in effect. This field applies only to files on the disk.
- 18** An asterisk indicates that the file was not closed properly. This field applies only to files on the disk.

The seven types of file-sharing are:

ISI The file is opened for input and will allow other OPEN statements for input.

ISH The file is opened for input and will allow other OPEN statements for input or output.

OSH The file is opened for output and will allow other OPEN statements for input.

OSI The file is opened for output and will allow other OPEN statements for input.

ISU The file is opened for input and will allow other OPEN statements for input or output.

OSU The file is opened for output and will allow other OPEN statements for input or output.

NS The file is open and will not allow others to share it.

For related information see:

“File sharing”

“PROTECT command”

“OPEN statement”

DIR

DIR command (continued)

“FREESP”

“Prepare diskette” under “Customer Support Functions”

DISPLAY

Display files and data

Display, as a type of data, refers to the input and output of data that can be printed or displayed. This includes transfer of data to or from devices such as the keyboard, display, printer, diskette and disk. While DISPLAY I/O may be performed to or from files, data is transferred in a format similar to that for display devices. The same format is used for BASIC source programs and procedure files on a diskette or on the disk.

Display files are accessed by:

- CLOSE (optional for diskette, disk, and system printer)
- INPUT
- LINPUT
- LOAD (SOURCE or DATA)
- OPEN (DISPLAY) (optional for the display and system printer)
- PROC
- PRINT
- REPLACE (SOURCE or DATA)
- RESTORE
- SAVE (SOURCE or DATA)
- SUBPROC

Example

This example directs program output to a selected device.

```
10 PRINT "Choose one:"
20 PRINT "  1  Printer output"
30 PRINT "  2  Feature printer"
40 PRINT "  3  Diskette output"
50 INPUT CHOICE
60 IF CHOICE = 1 THEN FILEID$ = "//10"
70 IF CHOICE = 2 THEN FILEID$ = "//11"
80 IF CHOICE = 3 THEN FILEID$ = "SAVE.REPORT//1, SIZE=0"
90 N$ = "NAME="
100 OPEN #1: N$&FILEID$,DISPLAY,OUTPUT
110 PRINT #1: "First line of report"
    .
    .
    .
500 CLOSE #1:
```

If the operator keys a 1, in response to INPUT statement 50, the output is directed to the system printer. If 2, the output is directed to the feature printer. If 3, the output is directed to the diskette.

DISPLY(X)

DISPLY(X)

DISPLY returns the value (1-5) of the current character group for the display. The X parameter, which is optional, is used to set the new page (see "Character set").

- 1 U.S.A
- 2 Canada
- 3 Europe, except Spain
- 4 Nordic, including Iceland
- 5 Spain, Spanish speaking countries

DROP/FREE command

The DROP command sets a file to the empty state, which sets the end of data pointer equal to the beginning of the file. The file space remains allocated.

The FREE command specifies that the file space reserved for the file is to be freed and may be allocated to another file. The file is no longer accessible after a FREE command.

For more information, see "Recover Diskette," "Recover Disk," under "Customer Support Functions," and "CLOSE statement."

Data is not modified on the file by either the DROP or FREE command. If security is required, the file may be written over by a BASIC program before you enter a DROP or FREE command.

DROP— file-spec —■

FREE— file-spec —■

The syntax of the DROP/FREE command is shown above, where:

file-spec is the filename, optionally followed by the VOLID, or device-id (see "File specification parameter") The file must not be opened when DROP or FREE is issued and must not be reserved by another station in a multi-station System/23.

Use FREE if you want to change the record length of an INTERNAL file.

Use the Copy Customer Support Function to compress unused space from the file (see "Customer Support Functions").

Any attempt to drop an index file will cause an error; use FREE.

Example

```
FREE FILEA
```

Dynamic file

Dynamic file extension

The creation of a System/23 type file requires the specification of an initial amount of space to be allocated to the file. The OPEN statement obtains this value from the SIZE parameter. (The SAVE command computes this value based upon the approximate size of the work area to be saved.) If at any time additional space is required because the original specification was too small, System/23 will automatically add additional space (called "extents" to the file (except as noted below and in "Diskette file size"). Each additional extent is 10% of the initial allocation rounded up to the next increment of 512. Up to 99 extents can be added to a file. If even more space is required, the file must be copied by the Copy Customer Support Function into fewer extents. For 5247 Disk files, allocations for space that fall adjacent to the last extent of the file are merged and a new extent is not created.

Note: If FORMAT=BX or HX is specified on an OPEN INTERNAL statement, the file created will not be extended beyond its initial allocation. BX and HX type files are invalid for 5247 Disk files. Z is the only format supported.

See also "DIR command" "Data buffering," "OPEN statement," "File size"

Editing a program or data file

Adding statements

You can add statements in the work area by entering a line number and a statement any time while your program is in the work area. The following may not be added while a program in execution is interrupted: DIM statement, OPTION statement, FOR statement, or NEXT statement.

Changing line numbers

You can change the line numbers in a BASIC program by entering the RENUM command. RENUM changes the line numbers to 00010, 00020, 00030, etc. See "RENUM."

Deleting statements

Enter DEL followed by a line number, or DEL followed by the first and last line numbers of consecutive statements. See "DEL command"

Replacing statements

You can replace one statement with another by entering the new statement with the old line number. You can enter it by editing the old line or entering a new line. The following statements cannot be changed while a program in execution is interrupted: DIM, OPTION, FOR, and NEXT.

several commands and BASIC statements can be entered by using the Cmd key and a special key. You should refer to your Keyboard Aids and "The keyboard" in your Operator Reference for a complete list of special keys.

Editing

Editing a program or data file (continued)

Programming considerations

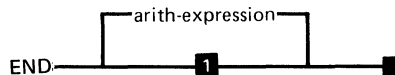
- Editing does not reduce the size of a program.
- If extensive editing is performed, the work area may fill up and an error will occur; save the program in source format and then load it. If a label is deleted, its absence will not be detected until it is referenced when the program is running.
- If the program exists on a file, do a REPLACE (see "REPLACE command").
- Editing the line following a GOSUB while a program in execution has been interrupted may cause unexpected results when program execution resumes.

ELSE

See "IF, THEN, ELSE statement."

END statement

The END statement specifies the end of a BASIC program and ends program execution. If the END statement is not specified, the system will still perform the END functions as if one were specified at the end of the program.



- 1** Code is set to 0

The syntax of the END statement is as shown, where:

arith-expression is the numeric value from 0 to 9999 which, after rounding, sets the CODE variable (see "CODE").

Execution of the END statement closes all open files and ends the program. The actions of the END statement are the same as those of the STOP statement.

Examples

```
910      END           !Value of CODE = 0
910      END 120       !Value of CODE = 120
```

Programming considerations

- Location
 - The END statement, if specified, must be the last statement of the program.
- CODE default
 - If the optional arith-expression is not specified, the default value of CODE is zero.

ERR

ERR is a system variable that contains the number of the most recently detected error (see "PROCERR command" and *System Messages*).

Error handling

Error handling

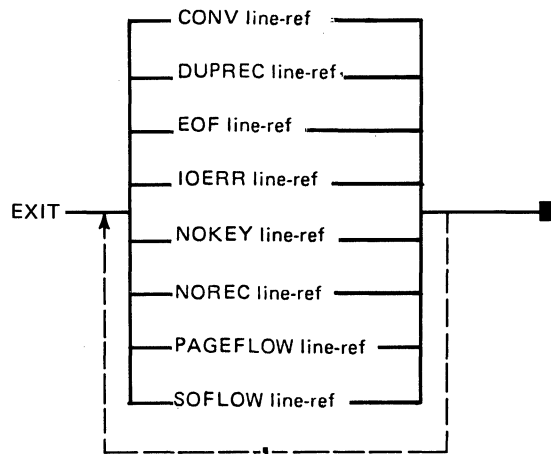
See "EXIT statement," "Interrupt," "System variables," "ON statement," *Customer Support Functions*, Volume I and Volume II, and *System Messages*.

Execution order

See "Order of execution."

EXIT statement

The EXIT statement specifies where control is transferred if an error occurs. The EXIT statement is descriptive and serves as a guide to the program. It indicates the line to which program control will transfer if an error occurs. The EXIT statement is referenced by an EXIT parameter on an input/output statement. When an error occurs, the EXIT statement is referenced. Program control will transfer to the line reference associated with the error condition.



The syntax of the EXIT statement is as shown above, where:

CONV, DUPREC, EOF, IOERR, NOKEY, NOREC, PAGEFLOW, and SOFLOW are error conditions for the various input/output statements.

CONV indicates a conversion error. There are four types of conversion errors:

EXIT

EXIT statement (continued)

- The I/O list item (numeric versus character) does not match the type of the FORM data conversion specification
- A numeric I/O list item will not fit within the field length specified in the FORM data conversion specification
- A numeric input field contains nonnumeric data
- A negative value is being output; the corresponding PIC data conversion specification does not indicate the sign.

DUPREC indicates that a record with the same relative record number already exists in the file referenced in the WRITE statement.

EOF indicates end of file.

- In a READ or INPUT statement, it means that there are no more records in the file.
- In a PRINT or WRITE statement, it means that there is not enough file space for the data.

IOERR for all input/output statements, this error indicates that an error has prevented completion of the statement which is not one of the other error conditions.

NOKEY indicates that no key matching the specified key can be found in the referenced file.

NOREC indicates that the specified relative record number is two or more greater than the relative number of the last record in the file or points to a deleted record.

PAGEFLOW indicates that the line printed is greater than or equal to (\geq) the *PAGEFLOW* value set in the *OPEN* statement (see "OPEN statement").

SOFLOW indicates that the number of input data characters is greater than the length of the I/O list character variable, or conversely, the length of the output I/O list character expression is greater than the field width defined in the *FORM* data conversion specification.

line-ref is a line number or a label symbol.

EXIT	Description	Value of ERR
CONV	Conversion error	0002 0726
SOFLOW	String overflow	0004 0058
DUPREC	Write to existing record	0054
NOKEY	No key found	4272
NOREC	No record found	0057
PAGEFLOW	Page overflow	0701
EOF	Not enough data items for READ	0054
	End of file	4270
	End of volume. No data space available to extend output file	4239
	No extents. Maximum number of extents assigned. Cannot extend output file	4271
IOERR	All other errors that occur on I/O statement	See Messages Manual

EXIT

EXIT statement (continued)

Example

An EXIT statement is shown below:

```
80 EXIT EOF 200, IOERR 220, NOKEY 240, NOREC 260
```

In this example, an input/output statement referencing line number 80 for the EXIT parameter causes program control to transfer to line number:

- 200 if an EOF condition caused the error
- 220 if an IOERR condition caused the error
- 240 if the key specified could not be found
- 260 if the record specified could not be found

Programming considerations

- Duplicate EXIT
 - Error conditions can be entered in any order. If a duplicate specification appears, it is ignored; the first one will be used.
- Arithmetic errors
 - Overflow, underflow, and zero divide conditions that are detected during the evaluation of an arithmetic expression cannot be trapped by exits on I/O statements. These conditions can be trapped using the ON statement.

Note: If an I/O list item is being mapped into a PIC data conversion specification, this rule is overridden. The overflow, underflow, or zero divide can be trapped by IOERR on the I/O statement or IOERR on the EXIT statement referenced by the I/O statement.

Example 1:

```
10 PRINT 1000*1.E+126 IOERR 20
20 END
```

Results in program interrupt 0003 on line 10
(9.999999999999999E+126 is displayed)

EXIT

EXIT statement (continued)

Example 2:

```
10 PRINT USING 20: .001*1.E-126 IOERR 30
20 FORM PIC(##. )
25 STOP
30 PRINT ERR
40 END
```

Results in a transfer to line 30.

- Data error
 - Overflow and underflow errors that are detected in data being read or entered can be trapped by IOERR on the I/O statement (or IOERR on the EXIT statement referenced by the I/O statement). If the exponent of the data item has more than three digits, this error is trapped by CONV.

Example 1:

```
10 INPUT A IOERR 20! where 1.E+130 is
15 STOP! the value entered
20 PRINT ERR
30 STOP
```

This example results in a transfer to line 20.

Example 2:

```
10 INPUT A CONV 20! where 1.E-9999 is
15 STOP! the value entered
20 PRINT ERR
30 STOP
```

Results in a transfer to line 20.

EXP(X)

Returns the value of e (2.71828182845905) raised to the X power. For example:

```
10 X=1
20 Y=EXP(X)
```

Statement 20 sets Y to 2.718281828459.

X >= 292 causes overflow.

X <= -292 yields an answer of zero.

Exponential

See "EXP(X)."

Expressions

Expressions

An expression in BASIC is a specification of a value using operators, constants, variables, arrays, array element references, and function references.

An arithmetic operator specifies an arithmetic operation to be performed on the data items.

Relational expressions are used with the IF statement to test the truth of specified relationships between two values. For example:

```
30 IF A>B THEN GOTO 100
```

Expressions referring to entire arrays, rather than individual array elements, are called array expressions. An expression that does not contain a reference to an entire array is called a scalar expression.

For more information on expressions, see:

- "Arithmetic expressions and operations"
- "Character expressions"
- "MAT assignment statements"
- "Relational expression" under "IF, THEN, ELSE statement"

Expressions, arithmetic

See "Arithmetic expressions and operations."

Expressions, array

See "MAT assignment statements."

Expressions, character

See "Character expressions."

Expressions, relational

See "Relational expression" under "IF, THEN, ELSE statement."

FIELDS

See "Full-screen processing."

FILENUM

FILENUM

FILENUM returns the numeric value of the file reference with the most recently detected error. -1 is returned if no errors have been detected.

Note: FILENUM must be assigned to a variable before printing or entering other I/O statement

Example

```
5 OPEN #2: "NAME=J",INTERNAL,INPUT
10 READ #2: A$ EOF 30
20 GOTO 10
30 A=FILENUM
40 PRINT "FILE"; A; "HAD AN ERROR"
50 CLOSE #A:
60 STOP
```

File reference parameter

The file reference parameter associates a logical file with a physical file or device when the file is opened. It is defined in the OPEN statement for the file and is then referenced by subsequent I/O statements that use the file or device.

File reference is an integer or numeric expression from 0 to 127 and 255. The integer or numeric expression must be preceded by a # (pound sign). System assigned file references not requiring an OPEN statement are:

0 Display, keyboard.

255 System printer.

I/O statements such as INPUT, LINPUT, and PRINT, when used to direct data to and from the keyboard or display, do not require a file reference parameter.

```
10 PRINT "HELLO"! Display message
20 PRINT #255: "TOTALS"! Print message
30 FILEID$="NAME=FILE"
40 OPEN #7:FILEID$,INTERNAL,INPUT! Open file
50 READ #7:A$ ! Read a record
60 CLOSE #7:! Close the file
```

File searches

File searches

When an OPEN (or implied OPEN) is executed, the file that is opened depends on the file specification and the actual location of the file.

- Filename only .. file

The search begins on drive 1 and continues on successively higher drive numbers. The first match is assumed to be the correct file. If other files of the same name exist on other disk volumes or diskettes, they are ignored. If not found, error 4000 is reported on the status line.

- Filename and drive ID .. file / drive

The search occurs on the specified drive only. If not found, error 4000 is reported on the status line if drive 1,2,3, or 4 was specified. Error 4152 is reported if drive 5 was specified.

- Filename and VOLID .. file / VOLID

The search begins on the lowest numbered drive. The first volume that is located with a matching VOLID is the only volume searched for the file. If the VOLID is found, but the file is not on that volume, error 4152 results. If the specified volume is not found, error 4000 results.

- Filename and VOLID and device .. file / VOLID / drive

The search occurs on the specified drive if the VOLID matches. If the VOLID matches and the file is not found, error 4152 is returned. If the VOLID does not match, error 4000 is reported on the status line if drive 1,2,3, or 4 was specified. Error 4152 is reported if drive 5 was specified. If the file / VOLID / device specification matches that of an open file, the *found* file must be marked OPEN for the new OPEN to succeed. That is, you cannot open two

different files with the same file/VOLID/device specification.

Note the following implications:

- If diskettes with the same VOLIDs are inserted, only the one in the lowest numbered drive is searched, if you do not specify a drive number.
- If offline data (see "Offline diskette files") is involved in the application, and open files exist on diskettes with the same filename and VOLID, unpredictable results can occur.
- The simplest and safest course is to code all file specifications with filename and VOLID and use unique filenames and VOLIDs.

File sharing

File sharing

File sharing lets you open a diskette or disk file two or more times simultaneously.

Within a single 5322 Computer, file sharing permits a program to use a file by two or more access methods, simultaneously. For example, you may wish to simultaneously access a file for sequential and direct input.

Within a System/23 consisting of two 5322 Computers and a 5246 Diskette Unit, file sharing permits two independently running programs in each 5322 Computer to share the same file in the 5246 Diskette Unit.

The two 5322 Computers can at any time independently access two different files with no restrictions, but the use of the 5246 Diskette Unit cannot be simultaneous. None of the subsequent discussion applies to this case (see "Device sharing").

If you have added a 5247 Disk Unit to your Sysstem/23, up to four computers may share a file.

File sharing is regulated by a set of OPEN parameters which specify what level of sharing is permitted by the other OPEN which has already, or will in the future attempt to use the file. When a conflicting use is detected, the second and subsequent invalid opens fail with a 4148 error.

The information required to perform this function is stored with the file. If an OPEN specifies restricted use of the file and no corresponding CLOSE is executed to terminate this restriction (power loss or diskette removal), then the PROTECT command must be used to cancel these restrictions. See "PROTECT command" For files on the 5247 Disk Unit, you do not need to cancel these restrictions unless you have coded a RESERVE parameter on the OPEN statement when you

opened a file. Use of the PROTECT command to CLOSE or RELEASE open files presently in use by a program should be avoided, as this can cause unpredictable results.

The default (no sharing specification), is no sharing permitted.

Sharing of Basic or H exchange files is permitted. No logical restrictions are imposed and any sharing other than input on both OPEN statements may produce unpredictable results. Any share specification is ignored for BX and HX files. Type Z is the only format supported for 5247 Disk files.

System/23 permits file sharing, but it is the responsibility of the programmer to see that the proper level of sharing restrictions are imposed to maintain data integrity. Furthermore, it is the joint responsibility of the application programmer and the operator to see that the System/23 is operated in a manner consistent with data integrity. This includes:

- Proper power sequencing
- Removal and insertion of diskettes at the correct time
- Proper use of system commands
- Proper execution of programs and procedures

File sharing is controlled by five parameters in the OPEN statement; they are SHR, SHRI, SHRU, NOSHR, and RESERVE. SHR means the other OPENS may use the file in any way, with the exception that only one OPEN may be for OUTPUT or OUTIN. SHRI means the other opens may do INPUT only. SHRU applies only to files on the 5247 Disk. SHRU means that up to four computers can open a file for OUTPUT, OUTIN, or INPUT. In addition, each computer can open the same file concurrently using a single program on that computer. NOSHR (default) means no other opens are permitted.

File sharing

File sharing (continued)

In summary:

First OPEN	Allowed subsequent OPENs
SHR,INPUT	SHRI,SHRU, or SHR, INPUT, OUTPUT or OUTIN
SHR,OUTPUT or OUTIN	SHR or SHRU,INPUT
SHRI,INPUT	SHRI or SHR, SHRU, INPUT
SHRI,OUTPUT or OUTIN	SHR,SHRU INPUT
NOSHR	none
SHRU,INPUT	SHR, SHRI, SHRU INPUT, OUTPUT, or OUTIN
SHRU, OUTPUT	SHR INPUT, SHRU INPUT, SHRU OUTPUT

The RESERVE option of the OPEN statement specifies that whatever sharing restriction is specified on this OPEN applies to the *other 5322 Computer* even after the file is CLOSED. This allows long term restriction of the file use, particularly when several programs, commands or Customer Support Functions must be run in succession without interference. The RESERVE option restriction does not apply to the 5322 Computer which issued it. Thus, even if a file is opened NOSHR,RESERVE, after the corresponding CLOSE is executed, any subsequent OPEN may be used by the same 5322 Computer.

The RESERVE status is cleared by the RELEASE option keyword on the CLOSE statement. The last program in a multi-step process would normally do a CLOSE...RELEASE to permit access to the file by the other 5322 Computer (this may be its only function). See "CLOSE statement."

The level of OPEN and RESERVE share restriction is indicated by the DIR DISPLAY. See "DIR command."

Programming considerations

- OPEN statements using key-indexed access (KEYED), place the same share restrictions on both the master and key files. (NAME= and KFNAME=).
- Share restrictions are ignored for BX and HX files.
- The RENAME, DROP, and FREE commands are rejected for a file which is open or has any reserve status set by the other 5322 Computer.
- The LINK, LOAD, MERGE, SORT, PROC, and SUBPROC commands open files INPUT, SHRI.
- The SAVE and REPLACE commands open files OUTPUT,NOSHR.
- The following Customer Support Functions ignore SHARE and RESERVE status, but secure the 5246 Diskette unit or the 5247 Disk Unit during their entire operation: Prepare Diskette, Copy (image copy only), Recover Diskette, Display Diskette Label, Prepare Disk, Recover Disk, and Add or Delete a Volume. Backup and Restore secures the volume only, not the entire disk unit. File Recovery will copy the SHARE and RESERVE status.
- Machine Update Generator, Collation Sequence Alternator, REPLACE, Presort, SORT and Index Generator open output files NOSHR.
- Presort, SORT, and Index Generator use WSID\$ as a suffix for work filenames.

File sharing

File sharing (continued)

- If both 5322 Computers load, edit and replace the same file, the last REPLACE will overlay any preceding REPLACE. To prevent this compromise of SOURCE files, open the file NOSHR,RESERVE and then CLOSE it. After editing and REPLACE, then PROTECT...RELEASE.
- The VOLID command can be used to change the disk volume or diskette VOLID at any time. If this is done while a file is open on the diskette or disk, it may prevent further processing of the file.
- If the Copy All Files option of the Copy Customer Support Function is used, the following errors are possible:
 - Use of the FREE, DROP, or RENAME commands during Copy can lose a file to be copied.
 - A file added to the input diskette while Copy is running may not be copied.
- Incorrect use of the PROTECT CLOSE or RELEASE options can compromise data integrity by removing share restrictions when they are still needed.

File size

The following can be used to estimate the storage used by various file types. See "OPEN statement."

<i>Type</i>	<i>Size (bytes)</i>
* BX	128 per record
* HX	256 per record
04	(1+RECL) per record
05	Total number of characters including blanks and new line characters
07	$512 * \text{CEIL}(\text{number of records} / (\text{INT}(512 / (\text{key length} + 4)) - 1))$
08	Value for type 07 plus $512 * \text{CEIL}(\text{number of new records} / \text{INT}(510 / (\text{key length} + 10)))$
09	(HELP STATUS size at CLEAR) - (HELP STATUS size when saved + (up to 2048 bytes)
81	(HELP STATUS size at CLEAR) - (HELP STATUS size when saved) + (up to 2048 bytes)
*	Invalid for 5247 Disk files

All files are automatically extended for additional output value

- Type BX and HX, which are fixed at OPEN to the SIZE value
- Files with no unallocated space; see "DIR command"
- Files with 100 extents; see "DIR command"

File size

File size (continued)

The following chart shows the maximum file size, in bytes, for each combination of diskette type and diskette format:

Diskette type	Diskette format		
	System/23	BX	HX
1	301,568	242,944	n/a
2	604,672	485,888	n/a
2D	1,135,104	n/a	985,088

- For 5247 Disk files, the file can be extended to the maximum capacity of the disk. When only one volume is active, maximum file size is:

30 M Disk = 30,271,488 bytes

15 M Disk = 14,848,000 bytes

See "File size."

File specification parameter

The file specification parameter consists of a filename, followed by a volume identification (VOLID) and device address. Filenames may be of the following types:

- Simple filenames may be from one to eight characters in length. The first character must be alphabetic (A–Z). The remaining characters may be alphabetic (A–Z) or numeric (0–9). Blanks are not permitted. Simple file names are required for Basic and H Exchange files.
- The names of the System/23 format (Z) files consist of one or more simple names separated by periods. The total number of characters, including periods, is 17.

```
CUSTOMER.EMPLOYEE  
X.Y.Z
```

VOLID identifies the disk volume or diskette on which the file is to be created or found. VOLID is up to six characters long and may consist of alphabetic or numeric characters.

Device address identifies the I/O device being used (see “Device address parameter”).

File specification can be in one of the following forms:

- filename
- filename/VOLID
- filename/VOLID/device
- filename//device
- //device

See “RENAME command” and “VOLID command.”

File specification

File specification parameter (continued)

Examples

```
CUSTOMER.EMPLOYEE//2  
X.Y.Z/TEMP  
//10
```

Note: The file specification //10 is used in an OPEN statement to open the system printer.

5247 Disk considerations

The 5247 Disk can store a large number of files. To save time in locating a file, use the most complete file specification parameter: *filename/VOLID/device address*.

If you do not specify the VOLID and device address, the computer searches all active volumes until it locates the file. The search sequence is:

```
device 1  diskette drive 1  
device 2  diskette drive 2  
device 3  diskette drive 3  
device 4  diskette drive 4  
device 5  disk unit...  volume 1  
                                     volume 2  
                                     volume 3  
                                     •  
                                     •  
                                     •  
                                     up to volume 25
```

Files, related subjects

CHAIN statement	LINPUT statement
CLEAR statement	ON statement
CLOSE statement	OPEN statement
CMDKEY	OPTION statement
CNT	PIC\$(C\$)
CODE	PRINT statement
Customer Support Functions	PROC command
DELETE statement	Procedure files
DIR	Device sharing
DISPLAY	PROTECT statement
DROP/FREE command	READ statement
END statement	REC(N)
ERR	Relative record files
EXIT statement	RELEASE statement
FILENUM	RENAME command
FILE(N)	REPLACE command
File reference parameter	REREAD statement
File sharing	RESERVE statement
File sizes	RESTORE statement
File specification parameter	REWRITE statement
FILE\$(N)	RLN(N)
FORM statement	SAVE command
FREESP(N)	SORT command
INPUT statement	STOP command
Internal I/O file formatting	SUBPROC command
I/O Tables (Appendix B)	USE command
Key-indexed files	VOLID command
KLN(N)	WRITE statement
KPS(N)	WSID\$
LINE	

FILE(N)

FILE(N)

FILE returns a numeric value to indicate the status of file N. One of the following values is returned:

<i>Value</i>	<i>Description</i>
-1	File not opened
0	Operation occurred successfully
10	End of file occurred during input
11	End of file occurred during output
20	Transmission error occurred during input
21	Transmission error occurred during output

FILE\$(N)

FILE\$ returns a string containing the file specification (file name, volume identification, and device address) of file N. If file N is not open, the null string is returned.

File types

The following table details the various file types processed by System/23.

File type	Contents/description	Created by	Record length	Input	Output	Access mode	Recoverable
BX(1)	Basic Exchange	OPEN INTERNAL Format BX	RECL 1 to 128	READ	WRITE	SEQ.	Yes (Note)
HX(1)	H Exchange	OPEN INTERNAL Format HX	RECL 1 to 256	READ	WRITE	SEQ.	Yes (Note)
04	Data	OPEN INTERNAL Format Z	RECL 1 to 4096	READ	WRITE	SEQ/REL KEYED	Yes (Note)
05	Data	OPEN DISPLAY SAVE SOURCE SAVE (data)	variable 0 to 255	PROC INPUT LINPUT LOAD [DATA]	PRINT REPLACE SAVE SOURCE	SEQ.	Yes (Note)
07	Index file without overflow area	Index GEN Create index file		READ KEY		KEYED	
08	Index file with overflow area	Index GEN OPEN KEYED OUTPUT		READ KEY	WRITE KEY	KEYED	No
09	Program file (internal)	SAVE		LOAD	SAVE REPLACE		No
81(2)	Program file (internal)	SAVE		LOAD			No
10	Customer Support Function	IBM		LINK			No
11	Feature	IBM		LINK			No
12	Machine update	IBM		SELECT UPDATE	SELECT		No
13	Diagnostics	IBM		CE diagnostic			No
14	Disk volume Backup	BACKUP RESTORE		RESTORE	BACKUP		No

Notes: To recover use Recover Diskette or Recover Disk Customer Support Functions.

1. Bx and Hx file types are not valid on the 5247 Disk.
2. Program contains RESERVE or RELEASE statements and may only be loaded on a computer when a 5247 Disk is attached.

Fixed point format

Fixed-point format

See "Arithmetic data."

Floating currency symbol

See "PIC specification" under "FORM statement"
"PIC\$(C\$)."

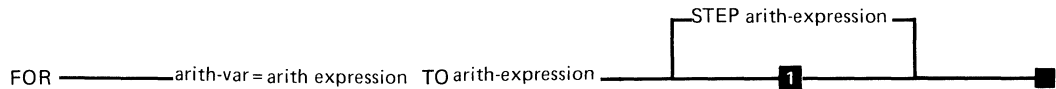
FNEND statement

See "DEF, FNEND statement."

FOR and NEXT statements

Together, a FOR statement and its paired NEXT statement delimit a FOR loop. A FOR loop is a set of BASIC statements that can be executed one or more times. The FOR statement marks the beginning of the loop and specifies the conditions of its execution and end. The NEXT statement marks the end of the loop.

FOR syntax



1 STEP=1

The syntax of the FOR statement is as shown above, where:

arith-var is an arithmetic variable (not an array name) used as the loop control variable and identify the associated NEXT.

arith-expression is an expression that specifies an initial value for the control variable, the final value of the control value (where execution of the loop will end), and the amount that the control variable will increment after each execution of the loop. If STEP and the increment-num are omitted, an increment of 1 is assumed.

Upon initial entry in the FOR loop, all expressions are evaluated. The initial value of the control variable is tested against the final value of the control variable. If the initial

FOR and NEXT

FOR and NEXT statements (continued)

value is greater than the final value for positive STEP values, or less than the final value for negative STEP values, the loop is not executed. In this case, the value of the control variable is set to the initial value and control goes to the statement following the NEXT statement. Otherwise, control goes to the statement following the FOR.

NEXT syntax

NEXT ——— arith-var ——— ■

The syntax for the NEXT statement is as shown above, where:

arith-var is an arithmetic variable used as the loop control variable. If the loop is executed, the control variable is set equal to the initial value, and the statements in the loop are executed. When the NEXT statement is executed, control is transferred to the associated FOR statement and the STEP value is added to the control variable, which is then compared with the final value. If the control variable for positive increments is less than or equal to the final value, the loop is executed again and the cycle continues until an increment is made that makes the control variable greater than the final value. At that time, control transfers to the first executable statement following the associated NEXT statement. If the increment is negative, the loop executes while the control variable is greater than, or equal to the final value.

Examples

The following example shows a simple FOR loop that increases the control variable A by 2 until the value of 25 is exceeded.

```
20 FOR A=1 TO 25 STEP 2
  .
  .
  .
90 NEXT A
```

The following example shows how to nest FOR loops. The internal loop is executed 100 times each time the loop is executed.

```
10 FOR J=A TO B STEP C
  .
  .
  .
150 FOR K=1 TO 100
  .
  .
  .
250 NEXT K
  .
  .
  .
300 NEXT J
```


FOR and NEXT statements (continued)

Programming considerations

- Parameters fixed at loop entry
 - The value of the control variable can be modified by statements within the FOR loop, but its initial value, its final value, and the STEP value are established during the initial execution of the FOR statement and are not affected by any statement within the FOR loop.
- Zero STEP
 - If the value of the STEP increment-num is zero, the FOR loop is executed until the value of the control variable is purposely set beyond the specified final value by a statement in the loop.
- Errors
 - Transfer of control into or out of a FOR loop is permitted; execution of a NEXT statement without execution of a corresponding FOR statement causes an error.
 - FOR loops can be nested within one another as long as the internal FOR loop falls entirely within the external FOR loop. Nested FOR loops should not use the same control variable, because the inner loop will modify the value of the outer loop control variable.

- The maximum number of nested FOR-NEXT loops is a variable number (usually around 50). If the maximum is exceeded, a system error occurs.
- Modification of a FOR or NEXT statement during execution is not permitted.
- Exit control value
 - The value of the control variable at exit from FOR/NEXT loop is the first unused value.

Example

```
10 FOR I=1 to 10
20 PRINT "TEST"
30 NEXT I
```

The value of I is 11

FORM

FORM statement

The FORM statement describes the way output should look when the PRINT, WRITE, or REWRITE statement is used. The FORM statement also describes the way input looks when using a READ or REREAD statement.

The FORM statement controls the number of output positions taken by a value being displayed. The following program writes `b b b99` in columns 1 through 5 on line 22 on the screen:

```
10 PRINT USING 20: 99
20 FORM N 5
```

where `N 5` is a data conversion specification. `N` specifies the format type *numeric*; `5` specifies the *field length*.

The FORM statement is also used to control the number of digits displayed in a decimal fraction. For example, the following program writes `12.35` in columns 1 through 5 on line 22 on the screen:

```
10 PRINT USING 20: 12.345
20 FORM N 5.2
```

where `N 5.2` is a data conversion specification. `N` specifies numeric; the field length is `5`, and the fraction is rounded to `2 decimal digits`.

To display character data, use the `C` data conversion specification. The following program writes "Number of parts in stock:" in positions 1 through 25 on line 22:

```
10 PRINT USING 20: "Number of parts in stock:"
20 FORM C 25
```

where C identifies the format type and 25 is the field length.

The following program shows two I/O list items being output:

```
10 PRINT USING 20: "Number of parts in stock:",99
20 FORM C 25,N 5
```

where C 25 is the data conversion specification for the first I/O list item and N 5 corresponds to the second. The output in columns 1 through 30 on line 22 is "Number of parts in stock: 99."

To increase the spacing between I/O list items, use the X data conversion specification. The following program outputs "Number of parts in stock: 99" in positions 1 through 35 on line 22:

```
10 PRINT USING 20: "Number of parts in stock:",99
20 FORM C 25,X 5,N 5
```

where X is the format type and 5 is the field length. X 5 causes five blanks to be inserted in the output. No I/O list item is associated with X 5.

The above examples show the FORM statement being used with the PRINT statement. The FORM statement can also be referenced by the READ, WRITE, REREAD, and REWRITE statements. When FORM is used with these statements, output is to a record in an internal I/O file, and input is from a record in an internal I/O file.

FORM

FORM statement (continued)

The following example shows three values being written into a record of an internal I/O file:

```
10 WRITE #n,USING 20: "XYZ",30,10
20 FORM C 3,N 4,N 4
```

The previous example assumes the record length of the records in the file is greater than or equal to 11. If the record length were less than 11, an error would occur, because the field length for the third I/O list item would span the end of the record. If this file were open for input, the second and third values in the same record could be read as follows: (The first value is skipped.)

```
10 READ #n,USING 20: A,B
20 FORM X 3,N 4,N 4
```

The following example shows an alternative way of reading the same values from the same record:

```
10 READ #n,USING 20: A,B
20 FORM X 3,2*N 4
```

where 2* is a replication factor. It says to use the N 4 data conversion specification twice.

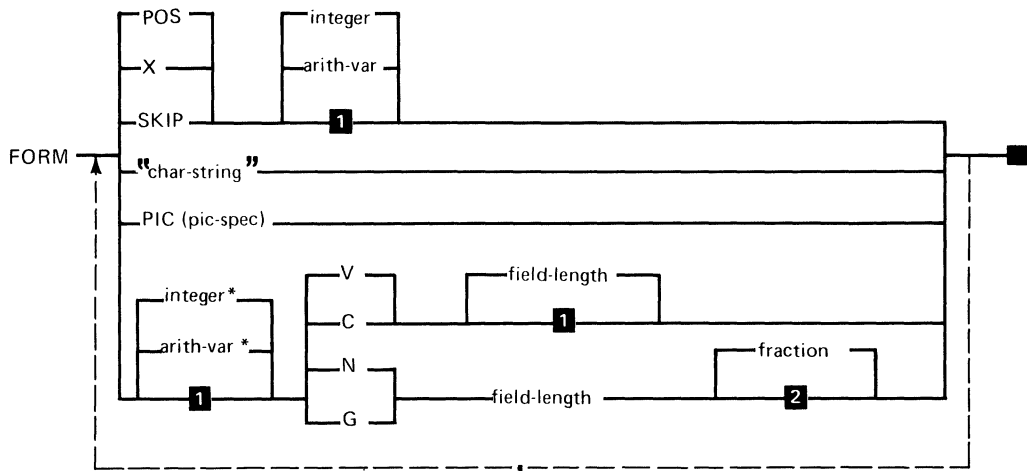
The FORM statement can be referenced by a line number or label in a USING clause of an I/O statement. The FORM statement can also be contained in a character variable. In the latter case the character variable is referenced in a USING clause in the I/O statement. Examples showing the FORM statement referenced by a label and the FORM statement contained in a character variable are included in the following program:

```

10 ! Label Reference
20 PRINT USING LAB1: 99
30 LAB1: FORM N 5
40 ! FORM Statement Defined in Character Variable
50 A$='FORM N 5'
60 PRINT USING A$: 99

```

Many additional data conversion specifications are supported by the FORM statement. The syntax of the FORM statement (when referenced by a PRINT statement) and the data conversion specifications, supported, are shown below:



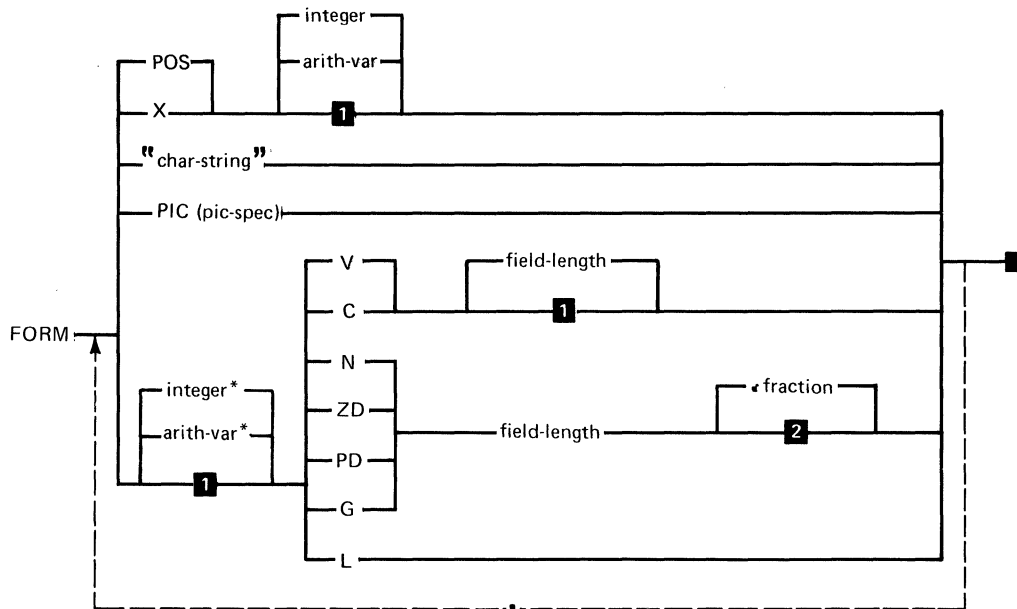
- 1** 1
- 2** Fraction length=0

Note that a blank is required between a format identifier (for example, POS or V) and any integer or variable which follows.

FORM

FORM statement (continued)

The syntax of the FORM statement (when referenced by a READ, WRITE, REREAD, or REWRITE statement) and which data conversion specifications are supported is shown below.



1 1

2 Fraction length=0

Note that a blank is required between a format identifier (for example, POS or V) and any integer or variable which follows. The "char-string" and PIC specifications are not supported on the READ and REREAD statements.

Detailed descriptions of the FORM data conversion specifications follow.

POS (for a PRINT statement) specifies the position in the line for the next value to be printed. If *POS* is less than the current position, the current line is printed and a new line started. The next I/O list item is printed in the new line at the position specified.

If one or more items have been printed on the current line and if the value of *POS* is beyond the end of the current line, positioning is as follows: Let *N* equal *POS* minus the current line position. The current line is then printed. *N* blanks are then written starting at the beginning of the next line.

The value specified for *POS* can range from 1 to 4095. The default is 1. Noninteger values in arithmetic variables are rounded.

POS (for a READ, REREAD, WRITE, or REWRITE statement) specifies the position in the record to be accessed. Positioning can be forwards or backwards in the record. The value specified for *POS* can range from 1 to the smaller of 4095 or the record length. The default is 1. Noninteger values in arithmetic variables are rounded.

Note: Output records are initialized to blanks by the WRITE statement and to the current record content by the REWRITE statement.

X (for a PRINT statement) specifies the number of blanks to be printed. If the value specified for *X* is greater than the number of positions remaining on the current line, the current line is printed, and the number of blanks specified for *X* is then written, starting at the beginning of the next line.

FORM statement (continued)

The value specified for *X* can range from 1 to 4095. The default is 1. Noninteger values in arithmetic variables are rounded.

X (for a READ, REREAD, WRITE, or REWRITE statement) specifies the number of positions to be skipped. The value specified for *X* can range from 1 to the smaller of 4095 or the number of positions remaining in the current record. The default is 1. Noninteger values in arithmetic variables are rounded.

Note: Output records are initialized to blanks by the WRITE statement and to the current record content by the REWRITE statement.

SKIP (for a PRINT statement) specifies that the current line is to be printed and that *n*-1 (where *n* is the value specified for *SKIP*) blank lines should appear in the output. The next output will begin in the first position of the following line. If the value specified for *SKIP* is zero, there will be no line feed, and overprinting will occur. See example at the end of this FORM statement section.

The value specified for *SKIP* can range from 0 to 255. The default value is one. Noninteger values in arithmetic variables are rounded.

char-string (for a PRINT, WRITE, or REWRITE statement) specifies a character string to be output. The field width is the length of the character string within quotation marks.

*integer** and *arith-var** specify the number of times the data format should be used. The same format can be used repeatedly. This parameter must range from 1 through 255. The default value is 1. Noninteger values in arithmetic variables are rounded.

C specifies character data. For a READ or REREAD statement, the number of characters specified by *C* are assigned from the input field to the character variable listed in the READ or REREAD statement. If the maximum variable length is less than the field-length specified, a string overflow (SOFLOW) occurs. If the variable length is greater than the fieldlength specified, the length of the character variable is set to field-length.

For input, an example is:

```
10 READ #n, USING 20
   :A$
20 FORM C 10
```

If the input field is

ABC **␣** **␣** **␣** **␣** **␣** **␣** **␣**

the trailing blanks are kept, and the variable A\$ is assigned a length of 10.

For output, the value of the corresponding character expression in the WRITE, REWRITE, or PRINT statement is left-justified in the output field and padded with blanks. If the length of the expression is longer than fieldlength, a string overflow (SOFLOW) will occur.

The *C* parameter is valid for character expressions and will cause a conversion (CONV) error if used with a numeric expression. The value specified for fieldlength can range from 1 to 255. The default is 1.

FORM statement (continued)

N specifies numeric data. For input, the number of record positions specified by the field-length must contain a numeric value in character form. The numeric value can have any one of the formats described in "Arithmetic data" (integer, fixed, or floating point). Leading and trailing blanks are ignored. If the numeric value is an integer, the number of digits specified by fraction length are used to generate the decimal fraction. The remaining high order digits in the field are used to generate the integer portion of the result. If the input numeric value is fixed or floating point, fraction length is ignored. For input, the numeric value is truncated to 15 significant digits. If option INVP is in effect, a comma in the input field will be treated as a decimal point. If the input field is left blank, a zero is the default.

For output, the corresponding numeric value in the output list is converted to character representation and is right-justified in the output field. If fraction length is not specified, the output field will contain the rounded integer value of the numeric expression. If fraction length is specified, the decimal fraction is rounded to the length specified. The result, including the decimal point, is placed in the output field. (If option INVP is in effect, a comma will be output in place of the decimal point.) If the numeric expression is negative, a minus sign will precede the numeric value in the output field. Plus signs are not inserted into the output field. The fieldlength must be large enough to contain any minus sign, integer digits, decimal point, and decimal digits.

The *N* parameter is valid for a numeric expression and will cause a conversion error if used with a character expression. The value specified for fieldlength can range from 1 to 26.

The following are examples of N format specifications:

Value to be written (decimal)	Specification	Resulting output (characters)
3.45	N7.2	3.45
3.45	N7.1	3.5
-3.45	N7	-3
-3.45	N7.1	-3.5

ZD specifies the zoned decimal format for numeric values. F or C is plus, and D is minus. No other values are allowed.

For input, the specification *ZD* field-length specifies that the next field-length bytes in the record contain a numeric value in zoned decimal form (one digit per byte). The optional specification, fraction length, identifies the number of rightmost digits to be used for decimal positions in the number. The default value is 0. See examples of *ZD* format specifications.

For output, an internal numeric value is converted to zoned decimal. If fraction length is not specified, the rounded integer value is used to generate the field. If fraction length is specified, the decimal fraction is rounded to the length specified. The field length must be large enough to contain all integer and decimal digits. The decimal point is not included.

FORM

FORM statement (continued)

The ZD parameter is valid for numeric expressions and causes a conversion (CONV) error if used with a character expression. The value specified for fieldlength can range from 1 to 32.

The following are examples of ZD format specifications:

Value to be written (decimal)	Specification	Resulting output (hexadecimal)
3.45	ZD7.2	F0 F0 F0 F0 F3 F4 F5
3.45	ZD7.1	F0 F0 F0 F0 F0 F3 F5
-3.45	ZD7	F0 F0 F0 F0 F0 F0 D3
-3.45	ZD7.1	F0 F0 F0 F0 F0 F3 D5

PD specifies the packed decimal format for numeric values. Fieldlength specifies the length of the field in bytes, and fraction length specifies the number of digits to the right of the decimal point. Each digit of a PD field occupies one half of a byte (4 bits), 2 digits per byte. The rightmost four bits are hexadecimal F or C for plus and hexadecimal D for minus.

For input, field-length specifies the number of bytes in a record containing a numeric value in packed decimal format (two digits per byte, with one digit and a sign in the rightmost byte). This value will be assigned to a numeric variable in a READ or REREAD statement. If the fraction length parameter is not specified, the field is assumed to contain an integer.

For output, field length specifies the number of record bytes into which the corresponding numeric expression from the WRITE or REWRITE statement is placed. The expression is converted to packed decimal format. If fraction length is not specified, the rounded integer value generates the field. If fraction length is specified, the decimal fraction is rounded to the length specified. The field length must be large enough to contain all integer and decimal digits, plus the sign.

The PD parameter is valid for numeric expressions and will cause a conversion (CONV) error if used with a character expression. The value specified for fieldlength can range from 1 to 32.

The following are examples of PD format specifications:

Value to be written (decimal)	Specification	Resulting output (hexadecimal)
3.45	PD.2	00 00 00 00 00 34 5F
3.45	PD.1	00 00 00 00 00 03 5F
-3.45	PD	00 00 00 00 00 00 3D
-3.45	PD.1	00 00 00 00 00 03 5D

L specifies internal, floating-point format (9 bytes) for numeric values.

For input, *L* specifies that an internal floating-point format value in the record is to be assigned to a corresponding numeric variable specified in the READ or REREAD statement. The contents of the field are not checked for validity.

FORM

FORM statement (continued)

For output, L specifies that the value of a numeric expression in the WRITE or REWRITE statement be written in the record in internal, floating-point format.

The following are examples of L format specifications:

Value to be written (decimal)	Specification	Resulting output (hexadecimal)
3.45	L	01 03 45 00 00 00 00 00
-3.45	L	01 83 45 00 00 00 00 00

V specifies variable length character data.

For input, fieldlength specifies the length of the field to be read. The string, excluding trailing blanks, is assigned to the character variable. The variable assumes that length. If the fieldlength is larger than the variable's maximum length, a string overflow (SOFLOW) occurs. An example of V-format is as follows:

```
10 READ #n, USING 20: A$  
20 FORM V 10
```

If the input field is ABCD **b b b b b**, trailing blanks are dropped and the data ABCD, with a length of 4, is assigned to the variable.

For output, the value of the corresponding character expression in the PRINT, WRITE, or REWRITE statement is left-justified in the output field defined by V and padded with blanks. If the length of the expression is larger than the field-width specified, a string overflow (SOFLOW) will occur.

The V parameter is valid for character expressions and will cause a conversion (CONV) error if used with a numeric expression. The value specified for field-length can range from 1 to 255. The default is 1.

G allows both character and numeric data to be used. If the I/O list item is numeric, the rules are the same as for N.

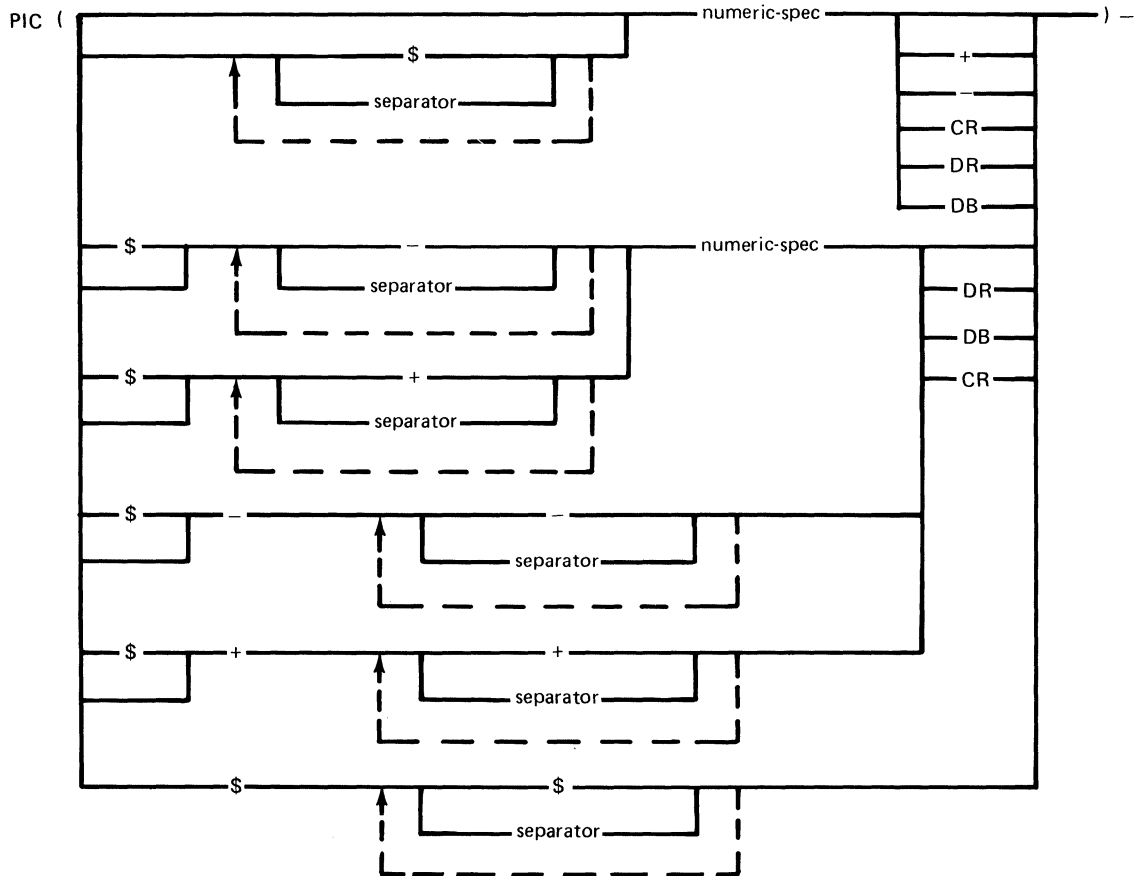
If the I/O list item is character, the rules are the same as for V. If the I/O list item is character, field length and fraction length are optional. If fraction length is specified, it is ignored.

FORM

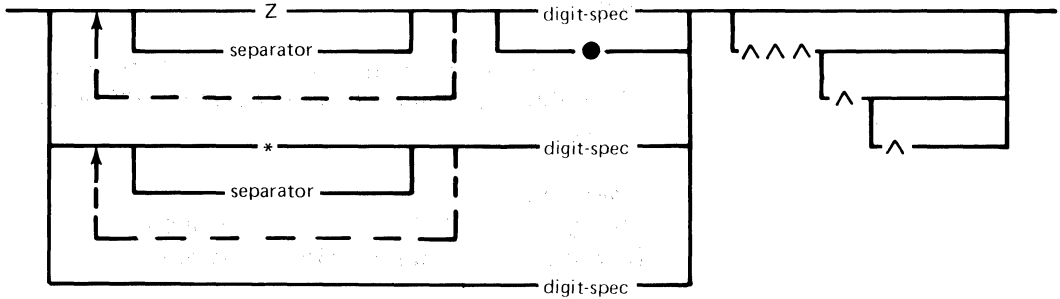
FORM statement (continued)

PIC specification

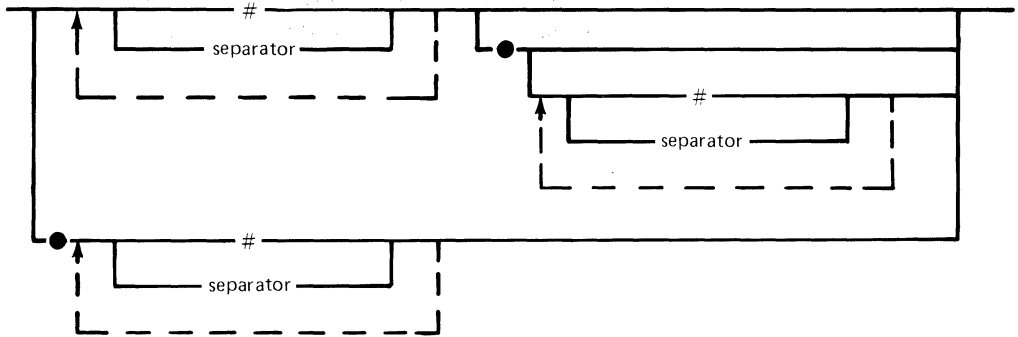
PIC is a data conversion specification having the following syntax:



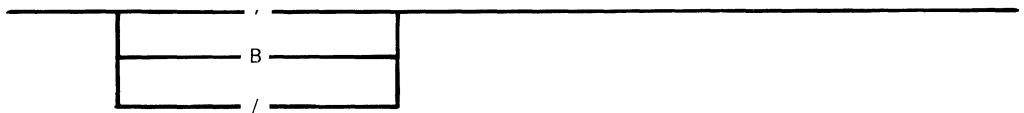
The syntax for *numeric-spec* is as shown:



The syntax for *digit-spec* is as shown:



The syntax for *separator* is as shown:



FORM

FORM statement (continued)

Each symbol represents one character position in the output. The output fieldwidth, (the number of symbols specified) can range from 1 to 32.

The I/O list item being output can be character or numeric. The following examples show the use of PIC to output character data:

Character string to be output	PIC specification	Printed output
August	PIC (#####)	August
May	PIC(#####)	May b b b
July	PIC(ZZZZZ)	July b
June	PIC(\$\$.##)	June b

Each # symbol represents one character to appear in the output. When the length of the character string is less than the fieldwidth, the character string is left-justified in the field and padded with blanks. When the character string length is greater than the fieldwidth, a string overflow (SOFLOW) will occur. When character data is being output, the # symbol and all other PIC symbols defined in the following paragraphs are character specifiers.

If the I/O list item is numeric, the PIC specification contains combinations of symbols which represent what the output should look like. The symbols are divided into the following four categories:

- Digit specifiers
- Insertion characters
- Exponent specifiers
- Trailing characters

FORM statement (continued)

Digit specifiers

The following digit specifiers can be used:

<i>Specifier</i>	<i>Meaning</i>
#	A numeric digit is printed.
Z	A numeric digit is printed. A blank replaces a leading zero (or conditional insertion character). Z may not appear to the right of a decimal point. Z is treated the same as a # if an exponent specifier is used.
*	A numeric digit is printed. An asterisk replaces a leading zero (or conditional insertion character). * may not appear to the right of a decimal point. * is treated the same as a # if an exponent specifier is used. For zero value the decimal point is replaced by an * if the decimal is the last character of the specification. * will not float across the decimal point to replace an insertion character.
\$	A dollar sign is printed. If more than one \$ symbol appears in the PIC specification, the currency symbol appears in the position of the rightmost \$ symbol which overlaps a leading zero (or conditional insertion character). The character to be printed as the currency symbol may be set by the PIC\$ function.

The default is \$. + or – may not precede the \$. \$ will not float across the decimal point to replace an insertion character \$ may not follow a decimal point. A specification of all \$, outputs zero as a single \$.

+ A plus sign is printed for a positive number, and a minus sign is printed for a negative number. If more than one + symbol appears in the PIC specification, the plus or minus sign will appear in the position of the rightmost + symbol which overlaps a leading zero. + may not precede a \$. A floating + may follow a single \$.

– A minus sign is printed for a negative number, and a blank is printed for a positive number. If more than one – symbol appears in the PIC specification, the minus sign or blank will appear in the position of the rightmost – symbol which overlaps a leading zero. A – sign may not precede a \$. A floating – may follow a single \$.

FORM

FORM statement (continued)

The following are digit specifier considerations.:

- A floating +, -, or \$ floats to the right across a B, comma, or a /. If the first significant digit is immediately to the right of a / or comma, the +, -, or \$ will replace the / or comma. A blank (B) is not replaced and the +, -, or \$ appears to the left of the B (blank).
- Although the System/23 permits Z or * to follow a floating \$, +, or -, this should be avoided since other systems may not support this function. For example:

PIC (\$\$\$**.#)

should be replaced by:

PIC (\$\$\$\$\$.##)

The following are examples of digit specifiers. Assume the data value 123456 is to be printed.

PIC specification	Printed output
PIC(#####)	000123456
PIC(ZZZZZZZZ)	123456
PIC(ZZZZZZ###)	123456
PIC(*****###)	***123456
PIC(\$\$\$\$\$###)	\$123456
PIC(+++++###)	+123456
PIC(----#####)	123456

If a dollar sign, plus sign, or minus sign is specified once in the PIC specification, it is printed in the position indicated.

PIC specification	Printed output
PIC(\$ZZZZZ###)	\$ b b 123456
PIC(+ZZZZZ###)	+ b b 123456
PIC(---#####)	123456
PIC(\$+++#####)	\$ b +123456

Using the value .05:

PIC(\$\$\$./##)	b b \$.05
PIC(\$\$\$B.##)	b b \$ b .05

Using the value 0

PIC(###)	b b \$
----------	----------------------

FORM statement (continued)

Insertion characters

Insertion characters add characters to a field, generally to improve readability. The following insertion characters can be specified:

<i>Character</i>	<i>Meaning</i>
B	A blank is printed.
,	A comma is printed. If a digit does not precede the comma, the comma is replaced by the zero suppression character (blank or asterisk) or dollar sign. If OPTION INVP is in effect, a decimal point will replace the comma in the output.
/	A slash is printed. If a digit does not precede the slash, the slash is replaced by the zero suppression character (blank or asterisk) or currency symbol.
.	A decimal point is printed. Only one decimal point may be specified. If option INVP is in effect, a comma replaces the decimal point in the output.

A PIC specification cannot begin or end with a B (blank), comma, or /.

PIC specification	Printed output
PIC(###B##B####)	000 b 11 b 2233
PIC(ZZZBZZBZ###)	11 b 2233
PIC(ZZZ,ZZZ,###)	112,233
PIC(ZZZZZ/Z#/##)	11/22/33
PIC(*****#.##)	*112233.00
PIC(\$\$\$,\$\$\$,\$\$\$.#)	\$112,233.00

FORM statement (continued)

Exponent specifier

The exponent specifier appears in the rightmost positions of a PIC specification, preceding the trailing characters, if any, as three, four, or five circumflex characters. The corresponding output positions are the letter E, the exponent sign (+ or -), and the exponent value. If the PIC specification also includes zero suppression symbols (Z, \$, +, -, or *), the # symbol is substituted for them. A decimal point always appears in the output in the same position as it appears in the PIC specification.

Values are rounded to the number of digit specifiers before output. For a floating field of +, -, or \$, the first specifier is not included in this number. All digits are used. The leading digit will be nonzero. An error occurs only if the exponent cannot be accommodated.

The following are examples of exponent specifiers. Assume a data value of 6.2345E+23 is to be printed:

PIC specification	Printed output
PIC(#####^ ^ ^ ^)	6234500E+17
PIC(##.###^ ^ ^ ^)	62.345E+022
PIC(##.##^ ^ ^ ^ ^)	62.35E+022
PIC(.#####^ ^ ^ ^ ^)	.623450E+24
PIC(ZZZ.##^ ^ ^ ^ ^)	623.45E+21
PIC(##.^ ^ ^ ^ ^)	62.E+022

Trailing characters

The following trailing characters can be specified:

<i>Character</i>	<i>Meaning</i>
+	A plus sign is printed for a positive number, and a minus sign is printed for a negative number.
-	A blank is printed for a positive number, and a minus sign is printed for a negative number.
CR, DB, or DR	The characters CR, DB, or DR, respectively, are printed for a negative number. For a positive number, either two blanks or two asterisks are output. Asterisks are output if * symbols were specified to suppress leading zeroes. Leading signs and CR, DB, or DR can appear simultaneously.

FORM

FORM statement (continued)

The following are examples of trailing characters. Assume that a data value of -123456 and 123456 are printed in alternating sequence.

PIC specification	Printed output
PIC (#####+)	0000123456-
PIC (#####+)	0000123456+
PIC (#####-)	0000123456-
PIC (#####-)	0000123456 ␣
PIC (ZZZZZZZZCR)	123456CR
PIC (ZZZZZZZZCR)	123456 ␣ ␣
PIC (*****DR)	***123456DR
PIC (*****DR)	***123456**
PIC (#####DB)	\$123456DB
PIC (#####DB)	\$123456 ␣ ␣
PIC (\$++++#####DB)	␣ -123456 ␣ ␣
PIC (\$++++#####DB)	␣ +123456DB

Programming considerations

- FORM statement
 - If the number of I/O list items exceeds the number of FORM data conversion specifications, the FORM statement is reused. (For a PRINT statement, there is a default SKIP 1 at the end of the FORM statement.)
 - Array elements are formatted in row order.
 - For a PRINT statement, if the field width specified in a data conversion specification is greater than the number of positions remaining on the current line, the current line is printed. The value to be output is then printed at the beginning of the next line.
 - FORM statements are nonexecutable and can be placed anywhere in a BASIC program; either before or after the I/O statements that reference them.
- PIC data conversion specification (numeric data)
 - The number of digit specifiers representing the integer portion of the value being output must equal or exceed the number of integer digits in the value itself. When one or more +, -, or \$ symbols are used as digit specifiers, one additional digit specifier is required.
 - The syntax of the PIC specification is not checked until the program runs.
 - The number of circumflex characters representing an exponent must equal or exceed the number of digits in the exponent itself, plus two.

FORM

FORM statement (continued)

- If a negative value is being output, the PIC specification must contain either a leading + or – specifier or a trailing +, –, CR, DB, or DR symbol.
- A PIC specification must contain at least one Z, *, #, or have at least two \$, +, or — specification characters.
- Values are rounded before output for fixed-point fields (no exponents). The value is rounded into the digit positions specified. For exponential output, the value is rounded to the number of digits specified. In both cases, one digit is deducted for floating \$, +, or –.
- If a floating currency string is followed by a decimal point, it must also be followed by one or more #.
- The floating + or – will not be printed for a zero value if no fractional digits are specified.
- If a floating currency string is followed by a trailing sign or exponent specifier, the currency field must be followed immediately by at least one #, Z, or *.

Example

The following program shows how to use the FORM statement to format printed output:

```
10 A=11
20 B$="ABC"
30 C=5
40 D$="DEF"
50 E=16.2
70 PRINT USING 80: A,B$,C,D$,E,"GHI"
80 FORM POS 3,N 3,X 2,C 4,SKIP 1,"COMMENT",
   PIC(###),V 4,2*G 4.1
```

The following output is displayed in lines 21 and 22, respectively.

```
   11ABC
COMMENT005DEF16.2GHI
```

Formatting I/O files

See "Internal I/O file formatting."

FREE command

See "DROP/FREE command" and "CLOSE statement."

FREESP(N)

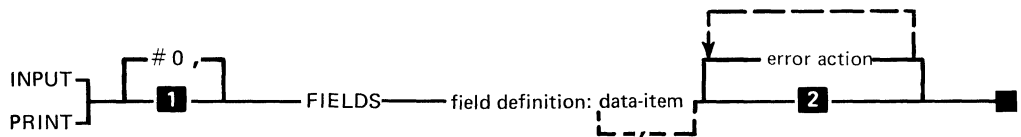
FREESP(N)

FREESP returns the number of 512-byte areas available for allocation on a disk volume or diskette that contains file N. Space allocation is made in 512-byte increments. A -1 is returned if the file is not open, if it is an exchange file, or if the device is not a diskette or disk.

Note: If file N is on a disk volume, this is the amount of space available on the entire disk, because disk volumes are dynamically extended until the disk is full.

Full-screen processing

Full-screen processing displays or enters data by letting you use the entire screen (except the status line). To display data with full-screen processing *PRINT FIELDS* must be entered. To enter data with full-screen processing, *INPUT FIELDS* must be entered. If the keyword *FIELDS* is not included, standard *PRINT* and *INPUT* processing is used. The syntax of the *PRINT/INPUT* statement for full-screen processing is as follows:



- 1** #0
- 2** Interrupt on error unless ON is active

where:

#0 is a numeric expression having the value of 0 for full-screen processing (see “File reference parameter”).

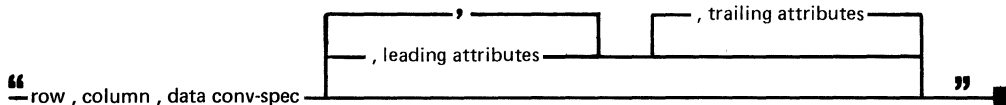
field definition is either a simple character variable or a MAT character array name (where character array name is a one-dimensional array). Avoid using substrings in the field definition.

data-item is a simple variable, subscripted array, or a MAT array name.

error-action is an EXIT line-ref or a CONV, SOFLOW, EOF, IOERR (see “EXIT statement”).

Full-screen processing (continued)

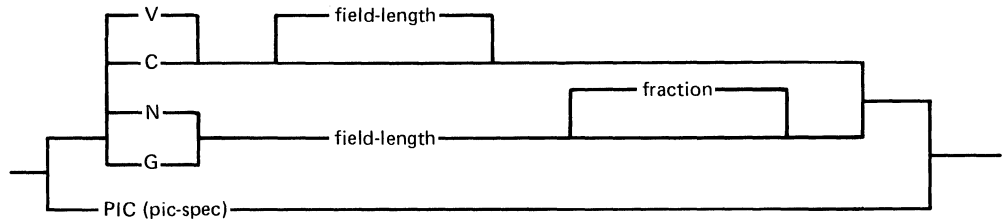
A character expression specified for FIELDS defines one field. Multiple fields are defined in a character array. The array element or character expression defining a field is a character string having the following syntax:



The parameters are positional and are delimited by commas. The insertion of blanks preceding or following individual parameters is permitted.

The starting location of a field is defined by the row and column parameters specified in the field definition. Row 1, column 1 is the upper left-hand corner of the screen. Fields may be defined on rows 1 through 23. The length of a field is the length specified either explicitly or implicitly in the data conversion specification. Fields may not span lines. The maximum length of a field is 78 (columns 2 through 79) for a field defined with leading and trailing display attributes. Leading and trailing display attributes are required for input fields. The maximum length of an output field without attributes is 80 (columns 1 through 80).

The following data conversion specifications are supported in a field definition:



V, C, N, G, and **PIC** have the same syntax and function as described under the "FORM statement." **PIC** is not supported on an **INPUT FIELDS** statement.

Leading and trailing attributes

Two types of attributes may be associated with fields: display attributes and control attributes.

Full-screen processing (continued)

Display attributes

Display attributes affect the visual characteristics of the display screen. The attribute "blink," for example, specified as a leading attribute for a field, causes the field to blink. The attribute "normal" specified as a trailing attribute, returns the screen to normal. Display attributes occupy screen locations. These locations appear blank. They may not simultaneously contain data. A display attribute affects the visual characteristics of the screen starting at the location following the attribute up through the location preceding the next display attribute.

The following display attributes are supported:

- I—Invisible
- U—Underline
- B—Blink
- H—Highlight
- R—Reverse (black on green)
- N—Normal (visible, no underline, no blink, no highlight, green on black)

When specified in combination, attribute I overrides attributes U, B, H, R, and N. Attributes U, B, H, and R override attribute N.

Example

Underline and blink:

```
10 PRINT FIELDS "10,10,C 40,UB,N": "HELLO"
```

Highlight:

```
20 PRINT FIELDS "10,10,C 20,H,N": "Hi"
```

Underline input fields:

```
30 INPUT FIELDS "10,10,C 20,U,N": A$
```

Control attributes

The second attribute type is the control attribute, which is used to modify input field operations. The following control attributes are effective when specified as leading attributes in an input field definition:

- **C**—Position the cursor to this input field first. If **C** is specified for more than one field, the cursor is positioned to the last field in the array having the **C** attribute.
- **A**—Automatic field exit. An automatic field exit occurs when the operator enters a character into the last location within this field.

Full-screen processing (continued)

- E—Automatic enter. An automatic enter occurs when the operator presses the Field Exit, Field Plus, or Field Minus key. An automatic enter also occurs when the operator enters a character into the last position of a field having the A attribute.

Control attributes do not occupy screen locations.

Any combination of display and/or control attributes may be specified for leading or trailing attributes in a field definition. Control attributes are inactive when specified for a PRINT field or as trailing attributes for an INPUT field. When an attribute is not recognized, it is ignored. If more than one display attribute is specified, the combination occupies one screen location. This location is R, C-1 for leading attributes and R, C+L for trailing attributes (R, C, and L are row, column, and length of the field). Input fields require both a leading and trailing display attribute. If either attribute is not explicitly specified in a field definition, the default for that attribute is N (normal). Output fields require neither leading nor trailing attributes; none are defaulted.

Examples

The following displays a field as defined by line 10.

```
10 A$="5,7,C 18"  
20 NAME$="JOHN DOE"  
30 PRINT NEWPAGE  
40 PRINT FIELDS A$: NAME$  
50 B$="8,2,C 10"  
60 INPUT FIELDS B$:DAK$
```

In this example, statement 40 displays the data on the fifth line of the screen, starting in column 7. The data item to be displayed is the character string "JOHN DOE". Statement 60 will input 10 characters from line 8, starting in column 2.

The following displays data on more than one line.

```
10 OPTION BASE 1  
20 REM DISPLAY NAME AND ADDRESS  
30 DIM FS$(3)  
40 NAME$="JOHN DOE"  
50 STREET$="125 1ST ST."  
60 CITY$="CHICAGO IL"  
70 FS$(1)="3,4,C 20"  
80 FS$(2)="4,4,C 25"  
90 FS$(3)="5,4,C 25"  
100 PRINT NEWPAGE  
110 PRINT FIELDS MAT FS$: NAME$, STREET$,CITY$
```

This example displays the first item of data (NAME\$) on the third line of the screen, starting in column four. The second item of data (STREET\$) is displayed on the fourth line of the screen, starting in column four, and so on.

Full-screen processing (continued)

The following displays data and enters data.

```
10 OPTION BASE 1
20 DIM A$(4), B$(3)
30 A$(1)="5,10,C 10,U,N"
40 A$(2)="10,4,C 5"
50 A$(3)="13,4,C 7"
60 A$(4)="16,4,C 5"
70 PRINT NEWPAGE
80 PRINT FIELDS MAT A$: "PROGRAMMER", "NAME:",
  "STREET:", "CITY:"
90 B$(1)="10,12,C 18,U,N"
100 B$(2)="13,12,C 18,U,N"
110 B$(3)="16,12,C 18,U,N"
120 INPUT FIELDS MAT B$: NAME$, STREET$, CITY$
```

See "Appendix A" programs 10, 11, and 12.

Programming considerations

- Number of fields
 - If an array is specified for FIELDS, the number of fields is the number of I/O list items, not the number of elements in the array. The number of elements in an array specified for FIELDS may exceed the number of I/O list items. The extra elements are ignored.
 - The maximum number of input fields is 128.
- Input attributes
 - When an INPUT FIELDS statement is executed, an implicit write to the display screen is generated to put out display attributes. The input fields are not modified.

- Order
 - The fields defined in a FIELDS array may appear in any order.
 - The first element or field in a FIELDS array corresponds to the first I/O list item. The second element or field corresponds to the second I/O list item, and so on.
- Enter
 - When the operator presses the Enter key, keyboard input ends and the input field values are processed. As each value is verified, it is assigned to an I/O list variable or array element. The number of I/O list items processed successfully is contained in the system variable CNT.
- Overlapped attributes
 - Input fields may not overlap. However, the location of the trailing display attribute of one field may be overlapped by the leading attribute of the following field. If leading and trailing attributes overlap, the last attribute written to the screen will be the one in effect.

Full-screen

Full-screen processing (continued)

- Mixed operations
 - Caution should be used when full-screen processing is interspersed with non-full-screen processing. There is no implicit clearing of the display screen when switching between the two. PRINT NEWPAGE may be used to perform this function. If the display screen is not cleared before full screen processing I/O, full-screen processing fields are interspersed with the previous contents of the screen. If the display screen is not cleared after full-screen processing I/O, data and/or display attributes left on line 23 may cause a syntax error on the next operator entry.

Functions, defined

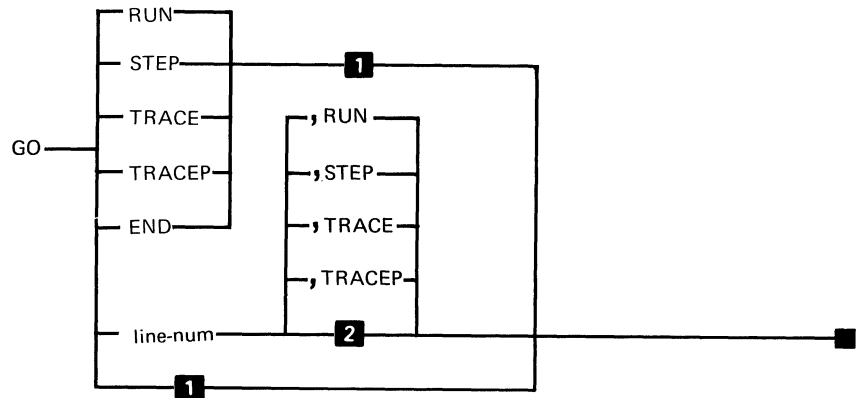
See "DEF, FEND statement."

GO command

The GO command resumes or ends processing of a BASIC program or procedure.

If a program or procedure is halted, processing can be resumed by entering GO. Program execution may continue with any line specified in the GO command.

Procedure execution may continue with the next procedure line. In order to continue execution with another line, see "SKIP command"



- 1** Resume execution at the current line of the program or the next procedure line.
- 2** Remain in previous mode.

The syntax of the GO command is as shown, where: *line-num* is the number of the line where processing of a program is to resume. If the line number is omitted, processing begins with the line logically following the last line executed or the next procedure statement.

GO

GO command (continued)

END specifies that all input and output files or the current procedure file are closed. After files are closed no program statements are executed (GO END is required for an interrupted program before issuing another RUN).

RUN specifies that processing is to continue in normal mode.

STEP specifies that processing is to continue in step mode (see "RUN command").

TRACE specifies that processing is to continue in trace mode (see "RUN command"). TRACE data is interspersed with screen data.

TRACEP specifies that trace messages are to be printed only. TRACEP should be used if tracing to screen would overwrite valid information.

Note: If neither RUN, STEP, TRACE, nor TRACEP is specified, processing continues in the mode that was in operation when processing was interrupted.

If a line number is not specified, RUN, STEP, TRACE, or TRACEP is not preceded by a comma (.).

Examples

To change to normal mode or resume normal operation of a BASIC program; type:

GO RUN (then press Enter)

To change to STEP mode and begin execution at line number 620; type:

GO 620, STEP (then press Enter)

Programming considerations

- Resume
 - GO may only be used to resume processing and not to initiate processing (see “RUN command” to begin processing).
- TRACEP printing
 - The data appears only when the line to be printed is full, the program generates a new line, or the printer is closed (when the program terminates).
- DISPLAY TRACEP
 - If the program is started by RUN DISPLAY, the TRACEP information is directed to the screen.

For more information see, “Split screen” and “TRACE statement.”

GOSUB

GOSUB and RETURN statement

The GOSUB and RETURN statements are used together to invoke subroutines. The GOSUB statement transfers control to a specified statement. The RETURN statement transfers control to the first executable statement following the GOSUB statement that invoked the subroutine in which the RETURN occurs.

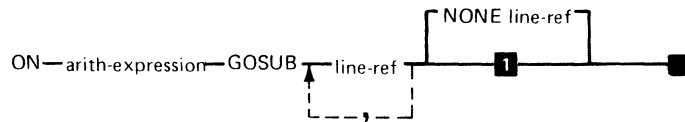
RETURN ————— ■

GOSUB ————— line-ref ————— ■

The syntax of the GOSUB statement is either simple or computed. The syntax of a simple GOSUB statement is shown above, where:

line-ref is the line number or label to which control is to be transferred.

Execution of a simple GOSUB statement transfers control to the line or label specified. The maximum nesting level is 200.



- 1** Interrupt occurs if the expression is out of range

The syntax of a computed GOSUB statement is shown above, where:

arith-expression is the arithmetic expression that determines the statement to which control is passed.

line-ref is a statement number or label. At least one statement number or label is required.

NONE if none of the line numbers preceding the *NONE* is selected, the line number following it is used.

Execution of a computed GOSUB statement causes the arithmetic expression to be evaluated. Control is then transferred to the line whose numeric position in the list of *line-num* (reading from left to right) is equal to the rounded integer value of the expression. Thus, an expression with a value of 2.75 would cause control to be transferred to the third line in the list. If the expression has a rounded integer value less than 1 or greater than the total number of lines listed, the program goes to the statement specified in the *NONE* clause. If a *NONE* clause is not specified, an error occurs.

When a GOSUB statement points to a descriptive statement such as DIM, control is transferred to the first executable statement following the descriptive statement.

GOSUB

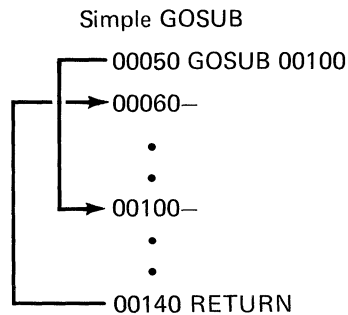
GOSUB and RETURN statement (continued)

Programming considerations

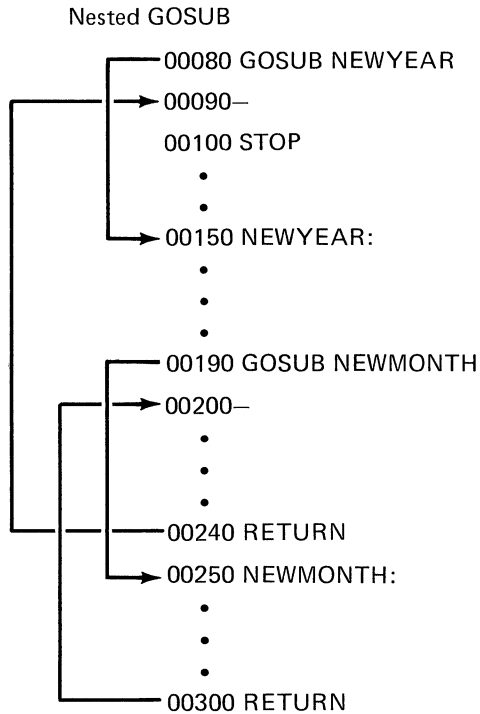
Subroutines should be preceded by a GOTO to avoid falling through into them.

Example

The following example shows the execution of GOSUB and RETURN statements:



Program 1—Line 50 transfers control to line 00100, stacking line 00060 as a return location. Assuming no further transfers, lines 00100 to 00140 are executed and line 00140 transfers control to line 00060.



Program 2—Assuming no transfer statements other than those shown, the order of execution is: 00080, 00150 to 00190, 00250 to 00300, 00200 to 00240, 00090 to 00100(STOP).

GOSUB

GOSUB and RETURN statement (continued)

An example of a computed GOSUB is as follows:

```
90 N=5  
100 ON (C/N) GOSUB 150,280,370 NONE 420
```

IF C is 5, line 150 gains control.

IF C is 10, line 280 gains control.

IF C is 15, line 370 gains control.

IF C is less than 2.5 or greater than or equal to 17.5, line 420 gains control.

GOTO statement

The GOTO statement transfers control to a specified line or label.

The syntax of the GOTO statement can be simple or computed.

GOTO — line-ref — ■

The syntax of the simple GOTO statement is shown above, where:

line-ref is the line number or label to which control is to be transferred.

Execution of a simple GOTO statement causes transfer of control to the line number or label specified.

The syntax of the computed GOTO statement is:

ON — arith-expression — GOTO — line ref — ■

↑ NONE line-ref

↑

, —

1 Interrupt occurs if expression is out of range

where:

arith-expression is the arithmetic expression that determines the line to which control is passed.

GOTO

GOTO statement (continued)

line-ref is a statement number or label. At least one statement number or label is required.

NONE if none of the line numbers preceding the *NONE* is selected, the line number following it is used.

Execution of a computed GOTO statement causes the arithmetic expression to be evaluated. Control is transferred to the line whose numeric position in the list of line numbers (reading from left to right) is equal to the rounded integer value of the expression. Thus, an expression with a value of 2.75 causes control to be transferred to the third line in the list. If the expression has a rounded integer value less than 1 or greater than the total number of lines listed, the program goes to the statement specified in the *NONE* clause. If no *NONE* clause is present, an error occurs.

When a GOTO statement points to a descriptive statement such as DIM, control is transferred to the first executable statement following the descriptive statement.

The following statement transfers control to line number 20:

```
100 GOTO 20
```

The following statement transfers control to the statement labeled RECOVERY when the variable LIMIT is 1.



```
100 ON LIMIT GOTO RECOVERY,500 NONE QUIT
```

Programming considerations

- If the variable in the expression has a subscript that is a numeric constant, the constant cannot have an odd number of digits ending in 27. If you do, results will be unpredictable when you issue a LIST command. For example, ON I(127) GOTO A,B will not list properly.

HELP STATUS command

The HELP STATUS command displays the amount of space (in bytes) available in the work area, the work area type (PROGRAM or DATA), and the file specification of the last file used to load or save the work area.

HELP STATUS  

The syntax of the HELP STATUS is as shown. Between CLEAR or power on and LOAD/SAVE/REPLACE, only the mode and number of bytes available are reported.

Example

```
HELP STATUS
20168          PROGRAM  PAYROLL.FDP/PAYROL
(bytes avail)  (type)   (file specification)
```

HEX\$(A\$)

HEX\$(A\$)

Returns a character string containing the hexadecimal value represented by the content of A\$. For example:

```
10 A$="F1F2"  
20 B$=HEX$(A$)
```

A\$ must contain only the digits 0 through 9 or the uppercase letters A through F. The number of hexadecimal characters must be even.

B\$ contains a two character string which is "12."

See "EBCDIC character table" under "Character set."

Hierarchy, arithmetic

See "Arithmetic hierarchy."

HOLD

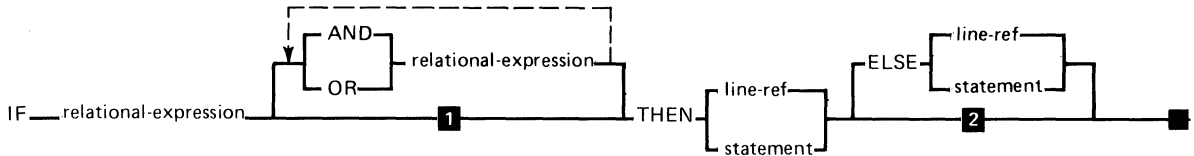
The HOLD key can be used to stop processing at any time (for example, to view the screen.) You press HOLD a second time to continue operation. Pressing the HOLD key does not immediately stop the printer.

If the 5322 Computer is sharing a 5246 Diskette Unit, the second 5322 Computer may be stopped also. See "Device sharing."

If diskettes are removed while the computer is in the hold state, unpredictable results may occur.

IF, THEN, ELSE statement

The IF, THEN, ELSE statement transfers control according to the result of an evaluated expression or conditionally executes a statement.



- 1** One expression considered
- 2** If expression is false, go to next statement

The syntax of the IF, THEN, ELSE statement is as shown above, where:

relational expression is a relational expression or a logical operator (see "Relational expression" and "Logical operators." under this (IF, THEN, ELSE) statement).

line-ref is the line or label to which control is to be transferred. It is specified by either a line number or a label symbol.

IF, THEN, ELSE

IF, THEN, ELSE statement (continued)

statement is any of the following BASIC statements:

CHAIN	MAT	REREAD
CLOSE	ON	RESERVE
CONTINUE	ON GOSUB	RESTORE
DELETE	ON GOTO	RETRY
GOSUB	OPEN	RETURN
GOTO	PAUSE	REWRITE
INPUT	PRINT	STOP
INPUT FIELDS	PRINT FIELDS	TRACE
LET	RANDOMIZE	WRITE
LET (implied)	READ	
LINPUT	RELEASE	

If CHAIN or MAT follows THEN, an ELSE clause is not allowed.

The IF, THEN, ELSE statement either transfers program control or executes a statement according to the results of a relational or logical expression. If the expression is true and a line reference follows the THEN, control is transferred to that line. If a statement is specified, instead of a line reference, that statement is executed. If the expression is false and a line reference follows the ELSE, control is transferred to that line reference. If a statement follows the ELSE, that statement is executed. If the execution of this statement does not result in the transfer of control, or the ELSE was not specified, then control is passed to the next executable statement in the program.

Relational expression

A relational expression compares the value of two arithmetic expressions or two character expressions. The expressions are evaluated and then compared according to the definition of the relational function specified. The relational functions and their definitions are:

Relational function	Definition
=	Equal
<> or ><	Not equal
=> or >=	Greater than or equal
=< or <=	Less than or equal
>	Greater than
<	Less than

When comparing numeric values, the value compared is the full 15 digits of the representation. Results of functions which are not accurate to 15 digits should be rounded before making an equal compare. See "Accuracy" under "Arithmetic data". When character data appears in a relational expression, it is evaluated according to the collating sequence, character by character, from left to right. When character operands of different lengths are compared, the result is unequal. If all characters of the shorter string are character-by-character equal to the leading characters of the longer string, the shorter string is less than the longer string. Blanks are significant in comparisons.

Example

```
10 IF A=B THEN 90 ELSE 110
```

IF, THEN, ELSE

IF, THEN, ELSE statement (continued)

Logical operators, expressions

Relational expressions are combined to form logical expressions.

Logical operators are used between relational expressions. When the logical operator AND is used between two relational expressions, the logical expression is true only if both relational expressions are satisfied. This is illustrated in the example that follows.

If OR is specified and the first expression is true, or if AND is specified and the first expression is false, the second expression will not be evaluated. For example, if the second expression contains a function, it will not be executed.

Examples

```
10 IF A$="JOB" AND B$="DATE" THEN 90 ELSE 110
20 IF MONTH=2 AND DAY=28 THEN MONTH=3 ELSE DAY=DAY+1
```

In the following example OR is used to specify that either of the two relational expressions can compare in order for the logical expression to be true.

```
10 IF A=e OR B<4 THEN 90 ELSE 110
```

The following is an example used for checking a blank field:

```
100 IF B$=RPT$(" ",LEN(B$)) THEN GOTO BLANK
.
.
.
500 BLANK:STOP
```

The following examples show a variety of IF statements:

```
30 IF A(3)<>X+2/Z THEN 100
40 IF R$="CAT" THEN 70
50 IF S2=37.222 THEN 120
60 IF X>Y THEN 90
70 IF A<B OR C<D THEN 110
80 IF A$="JOB" AND B$="DATE" THEN 100
90 IF A=3 OR B=4 THEN C=G ELSE STOP
100 STOP
```

In line 40, for example, if character variable R\$ contains the word CAT, program control is passed to line 70. In line 70, if either A<B or C<D, control is passed to line 110.

An example showing the use of labels is as follows:

```
30 IF MONTH=2 AND DAY=29 THEN LEAPYEAR ELSE LET
MONTH=MONTH+1
.
.
.
70 LEAPYEAR: LASTDAY=366
```

IF, THEN, ELSE

IF, THEN, ELSE statement (continued)

Programming considerations

- When an IF statement has a THEN clause and an ELSE clause, the THEN clause may not contain a MAT statement. For example, instead of doing this:

```
10 IF X=0 THEN MAT A=B ELSE MAT A=C
```

or:

```
10 IF X=0 THEN MAT A=B ELSE Y=10
```

Do this:

```
10 IF X=0 THEN MAT a+B
```

or:

```
10 IF X=0 THEN R=S ELSE MAT A=B
```

For other methods of examining data values, see "SRCH" and "POS."

Index keys

See "Create Index File" under "Customer Support Functions"

"DELETE statement"

"KLN"

"KPS"

"OPEN statement"

"READ statement"

"REREAD statement"

"RESTORE statement"

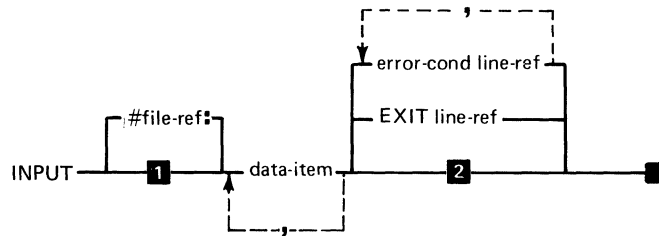
"REWRITE statement"

"WRITE statement."

INPUT

INPUT statement

The INPUT statement allows values to be assigned to variables from the keyboard (or procedure file) or a display file.



- 1** Defaults to #0
- 2** Interrupt on error unless ON is active

The syntax of the INPUT statement is as shown, where:

file-ref is a numeric expression, see "File reference parameter."

data-item is a simple variable, subscripted array element, or a MAT array name.

error-cond can be CONV, SOFLOW, EOF, IOERR (see "EXIT statement").

line-ref is either a line number or a label.

EXIT line-ref specifies the line number or label of an EXIT statement to refer to if an error occurs.

When an INPUT statement is executed, and input is expected from the keyboard, a question mark (?) is displayed on the screen on line 23, column 1 and the program execution halts. Input data is entered on the same line as the question mark. The data must be entered beginning in column 2 because column 1 is occupied by the question mark. You must then enter a list of values, which will be assigned in the order they are entered, to the variables listed in the INPUT statement or row-by-row to elements of specified arrays. The Enter key must be pressed to resume program execution. The number of values entered must be the same as the number of items in the I/O list.

INPUT is normally used to input data from the keyboard. However, it may also be used to read data (in keyboard entry format) from a DISPLAY file (type 05).

Assignment of values occurs after each ENTER or record delimiter. If the PROC option is entered in the RUN command, values are supplied from the active procedure file (see "Procedure file") rather than from the keyboard. Each INPUT statement gets one line from the display file or procedure file. If the record supplies too many or not enough values for the data list, an error is indicated.

Examples

To input:

A number or numbers, enter:

```
10 INPUT X
20 INPUT X,Y
```


INPUT

INPUT statement (continued)

A string or strings, enter:

```
10 INPUT N$
20 INPUT N$,A$
```

Numbers and strings, enter:

```
10 INPUT NAME$,AGE
20 INPUT X,X$
```

From a file, enter:

```
10 OPEN #100:"NAME=FILE.NAME",DISPLAY,INPUT
20 INPUT #100:ITEM1,ITEM2
```

An array (matrix), enter:

```
CLEAR
10 OPTION BASE 1
20 DIM ITEM$(3)
30 INPUT MAT ITEM$
40 PRINT ITEM$(1),ITEM$(2),ITEM$(3)
50 END
```

The operator may respond to the following INPUT statement:

```
10 INPUT NAME$,AGE,ADDRESS$
```

in either of the following ways:

? Gabe, 25, Street

or:

? Gabe,

? 25,

? Street

Programming Considerations

- Blanks
 - The only blanks allowed within a numeric field are leading blanks or trailing blanks.
 - Enclose a character field in quotes if leading blanks, trailing blanks, or delimiters are significant.
- Data items
 - The data types and the number of data items are verified before any assignment takes place.
 - The maximum length of each character data item entered, is 255.
- Procedures
 - Specifying PROC on the RUN command has no effect on input statements containing a file reference other than 0.

INPUT

INPUT statement (continued)

- Command keys
 - Command function keys, when pressed during INPUT, cause input to end (same as pressing the Enter key) and CMDKEY variable to be set.
- Cmd/Attn
 - Pressing the Cmd/Attn key while INPUT is pending, will cause an interrupt when the current INPUT statement completes execution (after pressing Enter).
- LINPUT
 - The unformatted input of a character string is achieved by using the LINPUT statement (see "LINPUT statement").
- EOF
 - Input from a procedure (RUN PROC) can cause an EOF condition at the end of a procedure. An EOF clause should be coded to account for this. The program cannot revert to keyboard input when started by RUN PROC.
- Terminating input with a slash
 - If the input data is terminated with a slash (/), the number of data items entered can be less than the number of I/O list items. The values of any remaining I/O list items are left unchanged.
- CNT
 - If the input data is terminated with a slash "/", only items preceding the slash "/" are counted.
 - Each data item is counted as one. *Example:*
100,200,300,400 (CNT=4)

- Null entries
 - When constructing a DISPLAY file (type 05) for processing by INPUT, and the last data item can be a null character string, end each line with a slash (/). This prevents an “ends in comma” error.
- Error conditions
 - Before values are assigned, a check is made of all the data values entered. If the check fails at any point, none of the entered values are assigned. Some potential errors which can occur are:
 - CONV means that character data was provided when numeric data was required.
 - SOFLOW means that the character string input was too long.

See “Program 5-Sample” in Appendix A and “Full-screen processing.”

Inquiry key

See “Attention and Inquiry.”

Integer format

See “Arithmetic data.”
“INT(X).”

Internal constants

Internal constants

An internal constant is a named, predefined value. Unlike arithmetic variables, the value is never altered during program execution. An example of the only internal constant is:

Constant	Name	Value
pi	PI	3.14159265358979

The internal constant name can only be used as a part of an arithmetic expression. It cannot be the target of an assignment statement. For example (assume rounding is to 7 digits), type:

```
PRINT 2*PI (then press Enter)
```

The result is 6.283185.

Internal files

See "Relative record files."
"Key-indexed files"
"Internal I/O files"

Internal I/O file formatting

Formatted (with USING)

When a WRITE or REWRITE statement contains a USING clause, the format of the data is specified by the associated FORM statement. The output record is generated in the following manner:

- Allocate a buffer of length specified by RECL= on the OPEN.
- Set the entire buffer to blank (hex 40). This applies to the WRITE statement only.
- Use the FORM specification and output data list values to fill in the specified record locations. Unspecified locations remain either blank or unchanged.

Internal I/O file formatting (continued)

Unformatted (without USING)

When a WRITE or REWRITE statement does not contain a USING clause, the record is "unformatted." The output record is generated as follows:

- Allocate a buffer of length specified by RECL= on the OPEN.
- In the first two bytes of the record, place the binary representation of the number of output list items. Each array element counts as one. The low-order byte is first, and the high-order byte is second.
- Preceding each data value place the binary representation of the length of the data item. Numeric items are length 9, and character items are specified by their current length (0 to 255).
- Following the item length, place the value of the data item in internal format. See "Arithmetic data" and "Character set."

The record length must have additional space allocated for these length fields over and above the aggregate length of the data. When numeric data items are expected, the length must be 9 bytes. When character data items are expected, any length is acceptable. No type checking is performed.

Internal I/O files

Internal I/O files are used for collecting related numeric and character data items and storing them as a unit in a fixed-length logical record. These files must be opened before using the WRITE (or REWRITE) statement to store data items in the file and the READ (or REREAD) statement to retrieve data items from the file. Internal I/O files can be accessed sequentially or directly either by key-indexed or by relative record number. For specific information, see "Relative record files" and "Key-indexed files." See also "REC(N)" and "RLN(N)."

Internal representation of characters

See "Character set."

Internal variables

Internal variables

See "System variables."

Interrupt

An interrupt is a condition that stops execution of the program or the procedure. After the interrupt occurs, the program or the procedure is allowed to continue. For more information, see:

"Attention and inquiry"

"EXIT statement"

"ON statement"

Interrupted programs may not be saved or replaced. DIM, OPTION, FOR, and NEXT statements may not be added or modified during an interrupt.

System Messages, SA34-0141, describes messages and tells how to fix each one.

Interrupt handling

BASIC program interrupts are:

- I/O errors
- Computational errors
- INQ key
- Cmd/Attn key

These interrupts are handled in the following priority:

- I/O errors with an applicable EXIT clause cause transfer to the specified line.
- Computational errors and I/O errors with no applicable EXIT clauses but with an applicable "ON condition GOTO" active, cause the specified transfer to take place.
- The INQ key is pressed and an ON ATTN GOTO is active. The INQ is detected prior to the execution of the next statement and the specified GOTO is executed. Note that Attn should be ignored when other conditions are being monitored by ON to prevent loss of one of the interrupts for RETRY and CONTINUE.
- Cmd/Attn cannot be intercepted and always causes an interrupt.

If no intercept (ON or EXIT) is specified for I/O errors, computational errors, or INQ key, an interrupt occurs.

For more information, see "ON statement" "Order of execution" and "Attention and inquiry."

For a description of special handling of ON events and I/O exits within a defined function, see "DEF, FNEND statement."

Intrinsic functions

Intrinsic functions

See "System functions."

INT(X)

Returns the largest integer not greater than X. For example:

```
10 X=-17.4  
20 Y=INT(X)
```

Y contains -18

```
10 X=3.4  
20 Y=INT(X)
```

Y contains 3

I/O action tables

The tables in Appendix B specify the response of System/23 to any combination of two I/O statements. Statements which are not listed on these tables are always considered errors. Refer to Appendix B for this information.

Keyboard

The keyboard is made up of alphabetic, numeric, and special character keys. Both uppercase and lowercase characters can be entered by using the shift key. The statement keywords and commands can be entered by using the Cmd key. When any of the keys are pressed, the characters entered appear on the input line on the display screen (see "Key description legend and tables").

Each of the keys are described in the *Operator Reference*, SA34-0108.

Keyboard

Keyboard (continued)

Programming considerations

Information that you enter (in response to the status line display of "READY INPUT") may change if it is not enclosed in quotes. This does not apply to input as a result of an INPUT or LINPUT statement or as a result of a data file. The changes are defined in terms of the EBCDIC character table shown under "Character set." Characters in a given column are changed to the corresponding character in the same row, with "column" determined by the following table:

Input character's column	Resultant character's column
0	4
1	5
2	6
3	7
4	4
5	5
6	6
7	7
8	C
9	D
A	E
B	F
C	C
D	D
E	E
F	F

For example, the *a* (in input character column 8) is changed to the *A* (in resultant character column C). The national currency symbols in column B are changed to the numerics in column F.

So, *PRINT 10¢* prints the number 100, whereas *PRINT "10¢"* prints 10¢. See "Character set," and "Variable names" for more information.

Keyboard-generated data files

A data file can be created directly from the keyboard by entering the CLEAR DATA command to clear the work area and define it as data. Next, the AUTO command may be entered to initiate automatic line numbering; or, lines may be entered preceded by a line number and a colon. The end of a data line is indicated when the Enter key is pressed. The only syntax restriction is the line number followed by a colon.

The length of the data file line may not exceed 249 characters. The work area can be saved with the SAVE command. Data file lines are saved without line numbers or colons. They are saved in the DISPLAY file type (05). Procedure files are an example.

When data file lines are listed, the colon is displayed. Data in a keyboard-generated file can be accessed as a DISPLAY file during program execution; or it may be a procedure file to control program execution. With line numbers and colons removed, the data file is accessed sequentially, one line at a time.

A saved data file can be changed by loading it back into the work area with a LOAD DATA command. When saved, the line numbers and colon are removed again. When loaded, data lines are preceded by line numbers, starting with 00010 and incrementing by 10, and the colon.

Key description

Key description legend and tables

The key description, legend used in conjunction with the key description tables, describes the action taken for each key on the System/23 keyboard.

Key description table legend

The following is a legend of the symbols used in the key description tables.

- A Display lowercase graphic associated with the key pressed
- B Display uppercase graphic associated with the key pressed
- C Display alternate shift graphic associated with the key pressed
- D Display the command shift keyword associated with the key pressed
- E Build a value between 0 and 255. The last three keystrokes between pressing and releasing the alternate shift key are used to build the value. Treat the value as follows:
 - $0 \leq \text{value} \leq 5$
 - Change the current CRT display page
 - $6 \leq \text{value} \leq 63$
 - Sound the audible alarm, ignore the value

- $64 \leq \text{value} \leq 255$
 - If the keyboard is open, display the EBCDIC character associated with the entered value; set KSTAT\$; else ignore.
- F Set KSTAT\$ to the value displayed.
- G Set KSTAT\$ to the value that would have been displayed if the keyboard were open.
- H Set KSTAT\$ to null.
- I Do nothing to KSTAT\$.
- J When key is pressed, set case shift state to uppercase, set to lowercase when released.
- K When key is pressed, set case shift to uppercase. Do nothing when released.
- L When key is pressed, set data shift to command, when released, set data shift as follows:
 - Alt (56) depressed (alternate)
 - Alt (56) not depressed (normal)
- M When key is pressed, set data shift as follows, when key is released, set data shift to normal.
- N Do not alter data shift state.
- O Terminate INPUT statement.
- P Set CMDKEY to zero.
- Q Set CMDKEY to 1-9, according to key pressed.
- R Scroll rows 1-23 down one logical line. The last logical line on the screen is lost. If listing a program, the preceding line of the program is displayed at the top of the screen if it will fit.
- S Scroll rows 1-23 up one line. If any part of the top logical line leaves the screen, blank the top logical line. If listing a

Key description

Key description legend and tables (continued)

program, the next line of the program is displayed at the bottom of the screen, if it will fit.

- T Enter ROS-resident diagnostics.
- Cmd/test causes error and action codes on status line.
 - Cmd/Attn = diagnostic monitor.
 - Error Reset = normal
- U Place machine in the hold state. All processing stops. Processing resumes when hold key is pressed again. Current I/O operations will run to completion. Copy Display is active in the hold state.
- V Copy the contents of the display screen, including the status line, to the system printer, device 10. Copy Display is active when the keyboard is open for input and when the machine is in the hold state.
- W When an error condition exists, fix the error successfully.
- X When an error condition exists, fix the error unsuccessfully.
- Y Set BASIC flag for "ON ATTN" condition. If program doesn't trap, same as Cmd/Attn
- Z Blank from the current cursor position to the end of the logical line. Scroll down the display screen so that the cursor is now on the entry row. The cursor is in the same position relative to the start of the logical line being operated on.
- a Blank field from current cursor position to end of field.
- b Return control to the system command processor. If BASIC is executing, control is returned before the next statement is executed. If a Customer Support Function or SORT is executing, control is returned at break points established by each function.

- c Move cursor to the first position of the next defined field. Cursor goes to first field if currently in the last field.
- d Move cursor to the last position of the previously defined field. Cursor goes to last field if currently in the first field.
- e If cursor is in a numeric field and field is not full, return an error. Otherwise, position cursor to row 23, column 1.
- f Move the cursor one position to the right (except for 'BE'-acute, '79'-grave, 'A1'-tilde, '5F'-circumflex, '9D'-cedilla, 'BD'-diaeresis).
- g Move the cursor one position to the left.
- h Wrap to the next row when leaving the right side of the screen. Wrap to the beginning of the logical line if leaving the right side of the screen on the last row of the line.
- i If the cursor leaves the field that it is in, move it to the first position of the next defined field. Move to first defined field if currently in the last defined field.
- j Wrap to the previous row when leaving the left side of the screen. Wrap to the end of the logical line if leaving the left side of the first row of the line.
- k If the cursor leaves the field that it currently is in, move it to the last position of the previous defined field. Move to the last defined field if leaving the first defined field.
- m Move the cursor to the first position of the previous defined field. Move to the last defined field if currently in the first defined field.
- n Wrap to the next row of the logical line if the cursor moves off the screen to the right. If there is no next row, scroll up the screen by one row and put the cursor at row 23, column 1. If the logical line would extend past 23 rows, sound the audible alarm and leave the cursor at row 23, column 80.

Key description

Key description legend and tables (continued)

- p If the field is Automatic Field Exit, then **i**. If the field is Automatic Field Exit and Automatic Enter, then **O** and **P**. If the field is not Automatic Field Exit and the cursor would normally leave the field, put the keyboard into the Field Exit Pending state.
- p Move the cursor one position to the right for each character displayed in the command keyword.
- r Move all the characters, from and above the cursor to the end of the line, one position to the right. Put a blank above the cursor. If nonblank data on the last row is shifted off the screen, extend the logical line. If the line is already 23 rows, return invalid key error.
- s Move all the characters, from and above the cursor to the end of the field, one position to the right. If the last position is nonblank before the operation, return an invalid key error. Put a blank above the cursor if the operation is successful.
- t Move all the characters from, but not above the cursor to the end of the line, one position to the left. Blank the last position in the line. If the entry row becomes blank, scroll down the display screen one row.
- u Move all the characters from, but not above, the cursor to the end of the field, one position to the left. Blank the last position in the field.
- v If the last position and the first position in the field are non-blank, return an invalid key error. If the first position is blank, put a minus sign there. If the first position is nonblank, shift the entire field right one position and put a minus sign in the first position.
- w If machine is in hold, enter C.E. monitor. If not, ignore.

Key description tables

These tables, used in conjunction with the key description legend, describe the action taken for each key on the System/23 keyboard.

Normal data keys: ~/ to \pm , Q to \;, A to {, \geq to ?/, space
 Key numbers: 1-13, 16-27, 30-41, 43-53, 57

Shift	Keyboard open for input		Keyboard closed
	Result for normal mode	Result for full-screen processing	Result
Lower-case (normal)	A, F, f, n	A, F, f, p	G
Upper-case	B, F, f, n	B, F, f, p	G
CMD key	D, q, n	D, q, n	N/A
ALT key	C, F, f, n	C, F, f, n	N/A

Key description

Key description legend and tables (continued)

± Copy D key
Key number: 13

Shift	Keyboard open for input		Keyboard closed
	Result for normal mode	Result for full-screen processing	Result
Lower-case (normal)	A, F, f, n	A, F, f, p	G
Upper-case	A, F, f, n	A, F, f, p	G
CMD key	V	V	V (press hold first)
ALT key	E	E	E

← key (Cursor Backspace)
Key number: 14

Shift	Keyboard open for input		Keyboard closed
	Result for normal mode	Result for full-screen processing	Result
Lower-case (normal)	g, j	g, k	N/A
Upper-case	g, j	g, k	N/A
CMD key	N/A	N/A	N/A
ALT key	N/A	N/A	N/A

Key description

← Key (Field Advance, Field Backspace)

Key number: 15

Shift	Keyboard open for input		Keyboard closed
	Result for normal mode	Result for full-screen processing	Result
Lower-case (normal)	O, P	c	N/A
Upper-case	O, P	m	N/A
CMO key	N/A	N/A	N/A
ALT key	N/A	N/A	N/A

Enter key

Key number: 28

Shift	Keyboard open for input		Keyboard closed
	Result for normal mode	Result for full-screen processing	Result
Lower-case (normal)	O, P	O, P, e	N/A
Upper-case	O, P	O, P, e	N/A
CMD key	N/A	N/A	N/A
ALT key	N/A	N/A	N/A

Key description

Key description legend and tables (continued)

↓ Key (Shift Lock)
Key number: 29

Shift	Keyboard open for input		Keyboard closed
	Result for normal mode	Result for full-screen processing	Result
Lower-case (normal)	K	K	K
Upper-case	K	K	K
CMD key	K	K	K
ALT key	K	K	K

↑ Keys (Upper Shift)
Key numbers: 42, 54

Shift	Keyboard open for input		Keyboard closed
	Result for normal mode	Result for full-screen processing	Result
Lower-case (normal)	J	J	J
Upper-case	J	J	J
CMD key	J	J	J
ALT key	J	J	J

← Key (New Line)
Key number: 55

Shift	Keyboard open for input		Keyboard closed
	Result for normal mode	Result for full-screen processing	Result
Lower-case (normal)	S	c	N/A
Upper-case	S	c	N/A
CMD key	N/A	N/A	N/A
ALT key	N/A	N/A	N/A

Alt key
Key number: 56

Shift	Keyboard open for input		Keyboard closed
	Result for Normal mode	Result for full-screen processing	Result
Lower-case (normal)	M	M	M
Upper-case	M	M	M
CMD key	N	N	N
ALT key	M	M	M

Key description

Key description legend and tables (continued)

Field

Exit key

Key number: 58

Shift	Keyboard open for input		Keyboard closed
	Result for normal mode	Result for full-screen processing	Result
Lower-case (normal)	O , P	a , c	N/A
Upper-case	O , P	a , c	N/A
CMD key	N/A	N/A	N/A
ALT key	N/A	N/A	N/A

Erase

Attn key

Key number: 59

Shift	Keyboard open for input		Keyboard closed
	Result for normal mode	Result for full-screen processing	Result
Lower-case (normal)	Z	a	N/A
Upper-case	Z	a	N/A
CMD key	b	b	b
ALT key	Z	a	N/A

Key description

Cmd key

Key number: 60

Shift	Keyboard open for input		Keyboard closed
	Result for normal mode	Result for full-screen processing	Result
Lower-case (normal)	L	L	L
Upper-case	L	L	L
CMD key	L	L	L
ALT key	L	L	L

Inq Key

Key number: 61

Shift	Keyboard open for input		Keyboard closed
	Result for normal mode	Result for full-screen processing	Result
Lower-case (normal)	Y	Y	Y
Upper-case	Y	Y	Y
CMD key	Y	Y	Y
ALT key	Y	Y	Y

Key description

Key description legend and tables (continued)

Hold key

Key number: 62

Shift	Keyboard open for input		Keyboard closed
	Result for normal mode	Result for full-screen processing	Result
Lower-case (normal)	U	U	U
Upper-case	U	U	U
CMD key	U	U	U
ALT key	U	U	U

Error

Reset key

Key number: 63

Shift	Keyboard open for input		Keyboard closed
	Result for normal mode	Result for full-screen processing	Result
Lower-case (normal)	N/A	N/A	W
Upper-case	N/A	N/A	W
CMD key	N/A	N/A	X
ALT key	N/A	N/A	W

Key description

Test key

Key number: 64

Shift	Keyboard open for input		Keyboard closed
	Result for normal mode	Result for full-screen processing	Result
Lower-case (normal)	W	W	W
Upper-case	W	W	W
CMD key	T	T	T
ALT key	W	W	W


 Key (Scroll Down)

Key number: 65

Shift	Keyboard open for input		Keyboard closed
	Result for normal mode	Result for full-screen processing	Result
Lower-case (normal)	R	c	N/A
Upper-case	R	c	N/A
CMD key	N/A	N/A	N/A
ALT key	N/A	N/A	N/A

Key description

Key description legend and tables (continued)

 Key (Scroll Up)

Key number: 66

Shift	Keyboard open for input		Keyboard closed
	Result for normal mode	Result for full-screen processing	Result
Lower-case (normal)	S	m	N/A
Upper-case	S	m	N/A
CMD key	N/A	N/A	N/A
ALT key	N/A	N/A	N/A



Del key

Key number: 67

Shift	Keyboard open for input		Keyboard closed
	Result for normal mode	Result for full-screen processing	Result
Lower-case (normal)	g, j	g, k	N/A
Upper-case	g, j	g, k	N/A
CMD key	t	u	N/A
ALT key	N/A	N/A	N/A



Ins key

Key number: 68

Shift	Keyboard open for input		Keyboard closed
	Result for normal mode	Result for full-screen processing	Result
Lower-case (normal)	f, h	f, i	N/A
Upper-case	f, h	f, i	N/A
CMD key	r	s	N/A
ALT key	N/A	N/A	N/A

1-9 (Ten Key Pad)

Key numbers: 71–73, 75–77, 79–81

Shift	Keyboard open for input		Keyboard closed
	Result for normal mode	Result for full-screen processing	Result
Lower-case (normal)	A, F, f, n	A, F, f, p	G
Upper-case	A, F, f, n	A, F, f, p	G
CMD key	O, Q	O, Q, e	G
ALT key	E	E	E

Key description

Key description legend and tables (continued)

Field

- key

Key number: 74

Shift	Keyboard open for input		Keyboard closed
	Result for normal mode	Result for full-screen processing	Result
Lower-case (normal)	N/A	a , v	N/A
Upper-case	N/A	a , v	N/A
CMD key	N/A	N/A	N/A
ALT key	N/A	N/A	N/A

Field

+ key

Key number: 78

Shift	Keyboard open for input		Keyboard closed
	Result for normal mode	Result for full-screen processing	Result
Lower-case (normal)	O , P	a , c	N/A
Upper-case	O , P	a , c	N/A
CMD key	N/A	N/A	N/A
ALT key	N/A	N/A	N/A

O key (Zero Key on the Key Pad)

Key number: 82

Shift	Keyboard open for input		Keyboard closed
	Result for normal mode	Result for full-screen processing	Result
Lower-case (normal)	A, F, f, n	A, F, f, p	G
Upper-case	B, F, f, n	B, F, f, p	G
CMD key	N/A	N/A	N/A
ALT key	E	E	E

• Key (Ten Key Pad)

Key number: 83

Shift	Keyboard open for input		Keyboard closed
	Result for normal mode	Result for full-screen processing	Result
Lower-case (normal)	A, F, f, n	A, F, f, p	G
Upper-case	B, F, f, n	B, F, f, p	G
CMD key	N/A	N/A	N/A
ALT key	N/A	N/A	N/A

Key indexed

Key-indexed files

A key-indexed file is an Internal I/O file with an associated index file. The index file cannot be used by itself; it must be used with a master file.

The index file is created by using the Create Index File Customer Support Function. You enter LINK INDEX to create the index.

The index includes a key, or set of up to 28 characters, that identify each record and the associated relative record numbers. An index file cannot be dropped with the DROP command.

Statements used to process a key-indexed file include:

Statement	Variation from Record I/O
OPEN	KFNAME=key filename, access method=KEYED
READ	KEY="identifying characters"
WRITE	do not specify KEY=
REWRITE	KEY="identifying characters"
DELETE	KEY="identifying characters"

More than one index can be created for the same master file. For example, with a master file of names and addresses, there could be two different index files:

- One with zip codes used as the key.
- One with names as the key.

For more information see:

- “OPEN statement”
- “READ statement”
- “WRITE statement”
- “REWRITE statement”

“DELETE statement”

Keys

See “Keyboard.”

KLN(N)

Returns the key length for file N. If the file is not open keyed, a -1 is returned.

KPS(N)

Returns the key starting position (byte number) for file N. If the file is not open as keyed, a -1 is returned.

KSTAT\$

KSTAT\$

Returns the character representing the key most recently pressed when the keyboard is not open. A reference to KSTAT\$ sets it to a null string.

See "Key description legend and tables."

Labels

Any BASIC statement (except a DEF statement) may be preceded by a label, which may be used in addition to a line number, to reference the line. The label is a one to eight character name with the same syntax as a numeric variable name (see "Line numbers" and "Line reference"). A label may not be the same as a variable name or a reserved word (see "Reserved words").

Example

```
10 START: GOTO IT
  •
  •
  •
500 IT:GOTO START ! ENDLESS LOOP
```

LEN(A\$)

Returns the number of characters in A\$, including blanks. For example:

```
10 DIM A$*100
20 A$="NUTS BOLTS SCREWS"
30 A=LEN(A$)
```

A contains the value 17

LET

LET statement (continued)

- Subscripted references to array elements are permitted in the assignment statement.
- Assignment of substring references are allowed.

Example

```
10 Z$ = "CAT"  
20 X = 9  
30 Y(X) = 2  
40 A$ = B$(4:5)  
50 A$ = A$ & B$(3:4)  
60 F$(X)(1:2) = B$(4:5)
```

After execution of line 10, the character variable Z\$ will contain the word CAT. In line 20, variable X receives a value of 9.

After execution of line 30, the tenth element (default is BASE 0) of the one-dimensional arithmetic array (Y) will have a value of 2.

After execution of line 40, the fourth and fifth characters of the character variable B\$ will be assigned to A\$.

After execution of line 50, the third and fourth characters of the character variable B\$ are appended to the end of A\$. A\$ is increased by two characters.

After execution of line 60, the first two characters of the tenth element in array F\$ are replaced by the fourth and fifth characters of B\$.

When line 10 is listed, it appears as:

```
00010 LET Z$ = "CAT"
```

LINE

LINE is the system variable containing the line number where the last BASIC program error occurred. If no error has occurred, it is zero. Nothing changes the value except another error.

See:

“CODE”

“ERR”

“Error handling”

“CONTINUE statement”

“RETRY statement.”

Line control

See “SKIP parameter” under “PRINT statement.”

Line numbers

Line numbers

Each line in a BASIC program and in a data file must begin with a unique line number. A line number is an integer from 1 to 99999 which tells the System/23 the line number in a program or data file. Lines do not have to be entered in line number sequence. They can be entered in any order since they are sorted by line number when they are stored. A line number must not be preceded by a blank. A BASIC program line number must be followed by a blank; a data file line number must be followed by a colon. For more information on line numbering, see "AUTO command" and "RENUM command."

Line reference

Each BASIC statement may be referred to in a program by either a label symbol (a name used to identify a line) or a line number.

Example

```
10 GO TO WORK
•
•
•
50 WORK: GO TO 10 !ENDLESS LOOP
```

LINK command

The LINK command loads and starts the Customer Support Functions, Communications Access Method (see *System/23 Communication Guide*), and the IBM System/23 Word Processing feature.

LINK— file-spec. — **1** — **■**

┌ , PROC ─┘

1 Input will be from the keyboard

The syntax of the LINK command is as shown above, where:

file-spec is the file name, which might optionally include the VOLID and device ID (see "File specification parameter").

PROC is used to allow certain programs to read input data from procedure file.

The system returns to CLEAR status when the linked program is completed.

When the LINK command executes, the system transfers control to the function specified.

LINK

LINK command (continued)

Example

A sample LINK command is as shown:

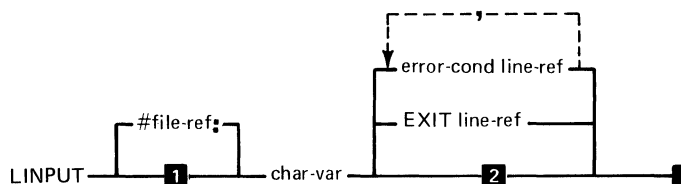
```
LINK COPY/VOL001
```

Upon execution of this command, the system will link to the COPY Customer Support Function on volume VOL001.

For more information on LINK, refer to "Using a procedure file" in Chapter 1 of *Customer Support Functions*, Volume II.

LINPUT statement

This statement permits unformatted input of a character string from the keyboard (or procedure file) or from a display file. The character string may contain commas, semicolons, leading blanks, and other characters which are delimiters in INPUT statement data. After removing trailing blanks, the maximum length string LINPUT will process is 255 characters.



- 1** Default is #0
- 2** Interrupt on error unless ON is active

The syntax of the LINPUT statement is as shown above, where:

file-ref is an integer or numeric variable from 0 to 127 and must be preceded by the symbol # (number sign). See "File-reference parameter."

char-var is a character variable into which the characters entered in the input line will be assigned.

EXIT line-ref specifies the line number or label of an EXIT statement to refer to if an error occurs.

error-cond line-ref can be CONV, SOFLOW (see "EXIT statement"). *line-ref* is either a line number or label.

LINPUT

LINPUT statement (continued)

LINPUT can access data written with a PRINT #n statement, but not with a WRITE statement. LINPUT can also be used for saved data files or programs which were saved in SOURCE format.

When Enter is pressed, keyboard input stops. LINPUT can also be ended by pressing the Cmd key and a number pad key. The CMDKEY variable will be set to the number pressed.

Specifying PROC on the RUN command has no effect on LINPUT statements containing a file reference other than zero.

Pressing Cmd/Attn will cause an interrupt when the LINPUT statement completes execution.

If the data for LINPUT is provided from a procedure file (RUN PROC) and data is no longer available, an EOF condition will occur. An EOF can be coded in the LINPUT statement to prevent an interrupt. The program cannot revert to keyboard input when started by RUN PROC.

Example

```
90 DIM ADDR$*255
100 PRINT "Enter full address"
110 LINPUT ADDR$
```

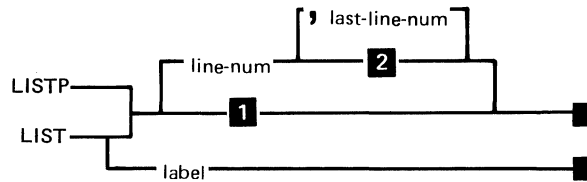
The full address can be entered as an answer.

See "Program 5—Sample" in Appendix A.

LIST, LISTP command

The LIST command displays the contents of the program or data in the work area on the screen. The contents of the work area are not changed.

The LISTP command prints the contents of the program or data in the work area on the system printer. The contents of the work area are not changed.



- 1** Screen—first 22 lines in work area
Printer—entire work area
- 2** Screen—22 lines preceding and including line-num
Printer—line-num to end of work area

The syntax of the LIST, LISTP command is as shown above, where:

label, when specified with LIST, causes the line with that to appear on the last line of the screen., The preceding line will be displayed also (as many as will fit on the screen).

line-num, when specified with LIST, causes that line to appear on the last line of the screen, with as many preceding lines as can be contained on the screen being displayed above that line.

LIST, LISTP command (continued)

line-num, when specified with LISTP, causes that line and all succeeding lines to be printed on the printer.

last-line-num specifies the last line number to be displayed or printed. It is used with *line-num*. Together, they specify that the range delineated by the two line numbers is to be printed or displayed. That is, all the statements with line numbers between *line-num* and *last line-num*, inclusive, will be displayed or printed.

If no line number or range is specified, a range consisting of the entire program is used.

Programming considerations

- Cmd/Attn keys
 - If the program is interrupted by Cmd/Attn or an error, LIST uses only the bottom five lines on the screen, while retaining the other display lines as they were (see "Split screen"). When execution resumes, the screen is returned to the status it had when interrupted.
- Nonexisting line numbers
 - If the line in the first *line-num* specification is not found, the next higher line number in the work area will be used.
 - If the line in the last *line-num* specification is not found, the next lower line number in the workarea will be used.
 - If the label does not exist, an error will occur.
- Long lines
 - If the line length exceeds the screen or printer width, the next line will contain the excess characters.

- Stopping LIST, LISTP
 - Since printer operations occur independently of other System/23 operations, the printer will not stop immediately when the Cmd/Attn or Hold keys are pressed. The printer will stop when all the information transferred to the printer has been printed.
 - The Cmd/Attn keys are recognized only on a listing operation to the printer.
- Internal editing
 - When programs are listed, unnecessary blanks and parentheses will be deleted; line numbers will be expanded to five characters and LET will be inserted for implicit LET statement. For example:

$$2 \ A = (B+C)$$

will list as:

$$00002 \ LET \ A = B+C$$
- Scroll
 - After LIST is issued, the screen may be scrolled up to display the succeeding lines in the work area. This can be done until either a line is entered or the Scroll Down key is pressed. Scroll Down will not display any preceding lines.
- Printer errors
 - Printer errors will not be reported until the next printer operation is attempted. The listing may be incomplete when a printing error occurs.

LIST,LISTP

LIST, LISTP command (continued)

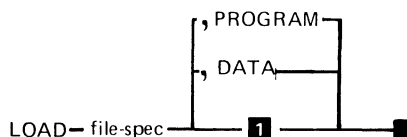
- Printing a program with the aid of LIST label
 - If you need a printed copy of the program, use LIST to list label to the screen to determine the line number of that label. Then, use LISTP and specify that line number.

Examples

- LIST
This will display the first group of work area lines that fit on the screen.
- LIST 250, 99999
This will display line number 250 and all succeeding lines that will fit on the screen.
- LIST 250
This will display line number 250 and all preceding lines that will fit on the screen.
- LISTP
This will print the entire program on the printer.
- LISTP 300, 500
This will print line number 300 through line number 500 on the printer.
- LIST 20,20
This will display one line number, line number 20.
- LIST ABC
This will display the line with the label ABC and all preceding lines that will fit on the screen.

LOAD command

Use the LOAD command to put a display file (type 05), a BASIC program, or the communications Customer Support Functions into the work area.



1 PROGRAM

The syntax is as shown above, where:

file-spec is the file specification parameter and is a filename followed by an optional VOLID and device address (see "File specification parameter"). File type must be 05 (display); 09 or 81 (BASIC program).

DATA specifies that the work area will be loaded from a file and that each line will be assigned a line number. The file specified must be a type 05. Line numbers will begin with 10 and increment by 10, for a maximum of 9999 lines.

PROGRAM is the default. If *DATA* is not specified, the file will be loaded as a program. No line numbers will be added since the program should already have line numbers. The file type must be 05, 09, or 81.

Example

```
LOAD PROG1//1
LOAD DISPLAY.FILE,DATA
```


LOAD

LOAD command (continued)

Programming considerations

- Cmd/Attn
 - A Cmd/Attn interrupt will not be honored during a LOAD operation.
- Insufficient storage
 - If insufficient space is available in storage to load the file specified, an error occurs. Enter CLEAR to restart operations. The program must be separated into two or more programs.
- Performance
 - A type 05 (display) file takes longer to load as a program than a type 09 or 81 (program) file, because in a type 05 or 81 file each line is syntax checked, while it appears on the screen.
- Closing Files
 - LOAD will close all files left open by an interrupted program.
- LOAD commands
 - If LOAD PROGRAM is issued to a data file that contains valid commands or calculator statements (without line numbers) they will be executed and LOAD will continue.
- Syntax errors
 - If a syntax error occurs, the system is put into error correction mode. The line in error can be manually corrected at this point. No other errors are recoverable. Instead of correcting a syntax error, the CLEAR command may be entered to terminate the loading. If the incorrect line is to be skipped, scroll down to the previous correct line and press Enter.

Logarithm

See "LOG."

Logical expressions

See "IF, THEN, ELSE statement"

LOG(X)

Returns the natural logarithm (base e) of X. If X is not greater than zero, an error is returned.

LPAD\$(C\$,X)

Returns a string of characters with a length greater than or equal to X by placing the required number of blanks before the first character of C\$. For example:

```
10 C$="ABCD"
20 A$=LPAD$(C$,5)
30 B$=LPAD$(C$,2)
   B$ contains "ABCD"
```

A\$ contains "ABCD"

Note: An error will be generated if X is not from 0 to 255.

LTRM\$(C\$)

LTRM\$(C\$)

Returns the string of characters contained in C\$. Leading blanks are removed. For example:

```
10 C$=" AB CD"  
20 A$=LTRM$(C$)
```

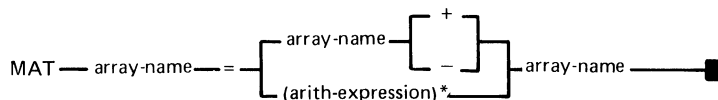
A\$ contains "AB CD"

Magnitude

See "Arithmetic data"

MAT assignment (addition, subtraction, scalar multiplication)

The MAT assignment statement (addition, subtraction, scalar multiplication) adds or subtracts the contents of two arrays and assigns the results to a third array. Or, the statement multiplies the elements of a numeric array by the value of an arithmetic expression and assigns the resulting products to the elements of another numeric array.



The syntax of the statement is as shown above, where:

array-name is the name of an array.

arith-expression is the value to be assigned. When the expression is evaluated, each element is set to that value.

The corresponding elements of the arrays specified to the right of the equal sign are operated on and assigned to the corresponding elements in the array specified to the left of the equal sign. To assign the elements of one array to another array, see "MAT assignment (simple)."

Programming considerations

- All arrays must be numeric.
- All arrays specified in the statement must have identical dimensions.

MAT

MAT assignment (addition, subtraction, scalar multiplication) (continued)

Example (addition and subtraction)

The following shows execution of a MAT assignment (addition and subtraction) statement:

```
10 DIM X(2,2), Y(2,2), Z(2,2)
```

```
•
```

```
•
```

```
•
```

```
100 MAT X=Y+Z
```

Each element of X now has the sum of the corresponding elements of Y and Z.

Example (scalar multiplication)

```
10 OPTION BASE 1
20 DIM X(2,2), Y(2,2)
  •
  •
  •
100 MAT Y=(4)*X
```

The expression (4) is evaluated. Each element in array X is multiplied by the value of the expression (4). The result is assigned to the corresponding elements of the array Y.

The resulting values are:

If X =

1	2
3	4

 then Y =

4	8
12	16

Each element of Y now contains four times the corresponding values in X. X is unchanged.

MAT assignment (ascending index or descending index)

See "AIDX and DIDX."

MAT

MAT assignment (scalar value)

This statement assigns a specified scalar value to each element of an array.

MAT `array-name=` $\left[\begin{array}{l} \text{(arith-expression)} \\ \text{(char-expression)} \end{array} \right]$ \blacksquare

The syntax of this statement is as shown above, where:

array-name is the name of the array that receives the values.

arith-expression is the scalar value to be assigned.

char-expression is the character value to be assigned.

Programming considerations

The expression to the right of the equal sign must be of the same type (arithmetic or character) as the array to which it is assigned.

Example

```
20 OPTION BASE 1
30 DIM Y(3,3)
40 MAT Y=(1)
```

The expression (1) is evaluated. Each element in the array Y is set to the value of the expression (1).

The resulting values are:

Y =

1	1	1
1	1	1
1	1	1

Y is now a 3x3 array with all elements equal to 1.

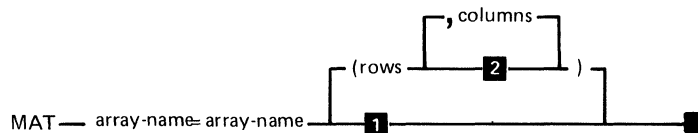
The following example causes all the elements of A\$ to be null:

```
10 MAT A$=("")
```


MAT

MAT assignment (simple)

This statement assigns the elements of one array to another array.



- 1 No redimensioning
- 2 One-dimensional

The syntax of this statement is as shown above, where:

array-name is the name of the array. The arrays specified must be the same type (numeric or character).

rows, columns are the redimensioning specifications for the receiving array (see "Redimensioning arrays").

Example

```
20 DIM A(2,2), B(2,2)
.
.
.
100 MAT A=B
```

Each element of array B is assigned to the corresponding element of array A.

The result of the above example is: A and B now have the same values in corresponding elements.

If redimensioning specifications are included with array B, the rounded integer portion of each expression value in rows and columns is used to redimension the receiving array A before values are assigned to it. Results are unpredictable if subscripted values are specified for rows and columns.

Programming considerations

- Redimensioning
 - If redimensioning specifications are included, rules described under “Redimensioning arrays” must be followed.
 - If redimensioning specifications are not included, arrays specified must have identical dimensions.
- Character arrays
 - If character arrays are used, the maximum character length of each must be the same.

Matrix

See “Arrays.”

Matrix

Matrix Operations

See "AIDX and DIDX"

"MAT assignment (addition, subtraction, scalar multiplication)"

"MAT assignment (scalar value)"

"MAT assignment (simple)"

"ZER and CON"

Maximum Value

See "Magnitude" under "Arithmetic data."

MAX (X1,X2,X3,...)

Returns the maximum value specified in the list. For example:

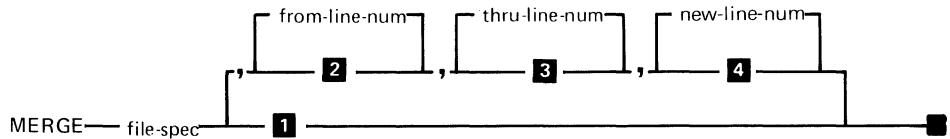
```
10 X=MAX(1,2,3,5,8,37,22,-21)
```

X contains 37.

MERGE command

The MERGE command is used to merge all or part of a saved file with data or with a program in the work area. In this way, the same routine can be added to several different files. Only BASIC statements (in a BASIC source file) and DISPLAY DATA files can be merged (file type 05) into the work area.

Lines from the file are added to the work area in line number sequence. If a line from the file and a line in the work area have the same line number, the line from the file replaces the work area line. If the merged file exceeds the size of the work area, a message will be displayed.



- 1** Merge all lines
- 2** Defaults to 1
- 3** Defaults to 99999
- 4** Must be specified for data.
May not be specified for program.

The syntax of the MERGE command is as shown, where:

file-spec is the name of the file to be merged into the work area (see "File specification parameter").

from-line-num is the number of the first line in the saved file to be merged. If no number is specified, the first line in the file is the default.

MERGE

MERGE command (continued)

thru-line-num is the number of the last line in the saved file to be merged. If no number is specified, the last line in the file is the default.

new-line-num is a parameter that is used only for DATA files and must be specified for DATA files. It is the first line number to be used in renumbering the saved file. If no number is specified, the merge will not take place. If a new line number is specified (for a data file), line numbering will be incremented by 10.

Omitted parameters must be indicated by consecutive commas.

Programming considerations

- Merging commands
 - If a program is merged and it contains commands or calculator statements, that do not have line numbers, the commands or statements be executed.
- Exceeding work area
 - If the merged file exceeds the work area, a message will be displayed.
 - If the work area is full, issue a CLEAR.
 - Deleting lines will not increase the available storage, unless a SAVE SOURCE command and a LOAD command are performed.
- Renumbering
 - If renumbering is necessary for a program file, the RENUM command must be used (see "RENUM command").

Example

```
MERGE PGM1,4,200
```

In this example, statements from PGM1 are merged with statements in the work area. Lines 4 through 200 will be merged.

```
MERGE DATAFILE,50,80,200
```

In this example, assume that the work area is defined as a DATA work area and that DATAFILE is a data file to be merged. The MERGE command logically associates line numbers (10, 20, 30, etc.) with the records in the data file. Line 50 is merged into the work area as line 200. Line 60 is merged into the work area as line 210. Line 70 is merged into the work area as 220, and line 80 is merged into the work area as line 230.

Minimum value

See "Magnitude" under "Arithmetic data."

MIN (X1,X2,X3,...)

Returns the minimum value specified in the list. For example:

```
10 X=MIN(1,2,3,5,8,37,22,-21)
```

X contains -21.

Multiple line

Multiple line function

See "DEF, FNEND statement."

Names, variable

See "Variable names."

NEXT statement

See "FOR and NEXT statement."

Numeric data formats

See "Arithmetic data." "FORM statement" (N specification).

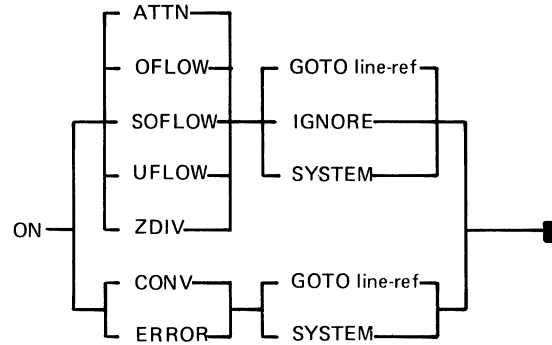
Offline diskette files

If an application requires that files be open on more diskettes than can be inserted in the number of available drives, the application can still function correctly if the operator is present to insert the required diskette when the System/23 calls for it. This task is greatly simplified if all involved diskettes have unique VOLIDs. Each time the program calls for a file, the System/23 may already have the required data in storage or on an inserted diskette. If the program requires data from a diskette that has been removed, and the file is already open, the status line shows error code 4001 and action code 10. Remove the diskette from the specified drive (if one is inserted) and insert the required diskette; press Error Reset. If the file is not already open, error code 4000 and action code 10 are displayed. If a drive is specified, remove the diskette from that drive. If no drive is specified, select a drive and remove the diskette from it. Insert the required diskette and press Error Reset. In either case, if for any reason you cannot provide the required diskette, press Cmd/Error Reset and the error will be returned to the program.

ON

ON statement

The ON statement causes the system to take a particular action when the specified condition occurs during execution of a program.



The syntax of the ON statement is as shown above, where:

ATTN is the condition associated with the depression of the INQ key. Control is passed following execution of the line in which the INQ key was pressed. ERR is set to 1 (see "Attention and inquiry."). For special considerations within defined functions, see "DEF, FNEND statement."

Note: Once the ON ATTN (inquiry) condition has been handled, CONTINUE will cause the program to resume at the point of interrupt.

OFLOW is the condition of numeric overflow. For example, an OFLOW error will occur when the system computes a number having an absolute value greater than the largest System/23 numeric value (see "Magnitude" under "Arithmetic data"). The result is replaced with plus or minus the largest System/23 numeric value.

SOFLOW is the condition of string overflow. For example, a *SOFLOW* will occur when there are more characters in the string than the variable has been dimensioned to hold, or when the variable has more characters than the associated *FORM* specification. If *IGNORE* is specified, the string is truncated on the right. If *SOFLOW IGNORE* occurs on a substring assignment or concatenation the error action is undefined; therefore, no assignment will occur.

UFLOW is the condition of numeric underflow. For example, a *UFLOW* error will occur when the system computes a number having an absolute value less than the smallest System/23 numeric value (see "Magnitude" under "Arithmetic data"). The result is replaced with zero.

ZDIV is the condition of division of a nonzero value by zero. The result is replaced with plus or minus the largest System/23 numeric value (see "Magnitude" under "Arithmetic data").

ON

ON statement (continued)

CONV is the failure to map data or change representation. For example, a *CONV* error will occur when the program tries to put alphabetic characters into a numeric field.

ERROR applies to those errors not covered by any of the above clauses (for example, I/O errors). *ERROR* also applies to any of the above for which an *ON* statement clause is not specified.

GOTO specifies that control is to be passed to the line specified by the line number or label symbol.

IGNORE specifies that the condition is to be ignored and control is to be passed to the next executable statement.

SYSTEM specifies that the condition is to cause an error.

Programming considerations

- EXIT
 - I/O statements having error exit parameters generally override the ON statement (see “EXIT statement”).
- ERR
 - If an error causes a transfer of control due to an ON statement, ERR is set and may be referenced.
- Error routine
 - To prevent repeated entry into the error routine, the error routine should begin with an ON condition SYSTEM statement.
- CONTINUE and RETRY
 - The statement CONTINUE will return control to the point following the interruption (see “CONTINUE statement”).
 - The statement RETRY will re-run the statement causing the interruption (see “RETRY statement”).

For more information, see “Order of execution,” “Interrupt handling,” “EXIT statement,” “Attention and Inquiry,” and “DEF, FNEND statement.”

ON

ON statement (continued)

Example

```
10 ON CONV GOTO ERR1
  •
  •
500 ERR1:PRINT "Conversion error,reenter data"
510 RETRY
```

OPEN statement

OPEN DISPLAY Activates a file for input; or, activates a printer or file for output.

OPEN INTERNAL Permits **READ**, **REREAD**, **WRITE**, **REWRITE**, and **RESTORE** statements to reference the file.

The **OPEN** statement is used to:

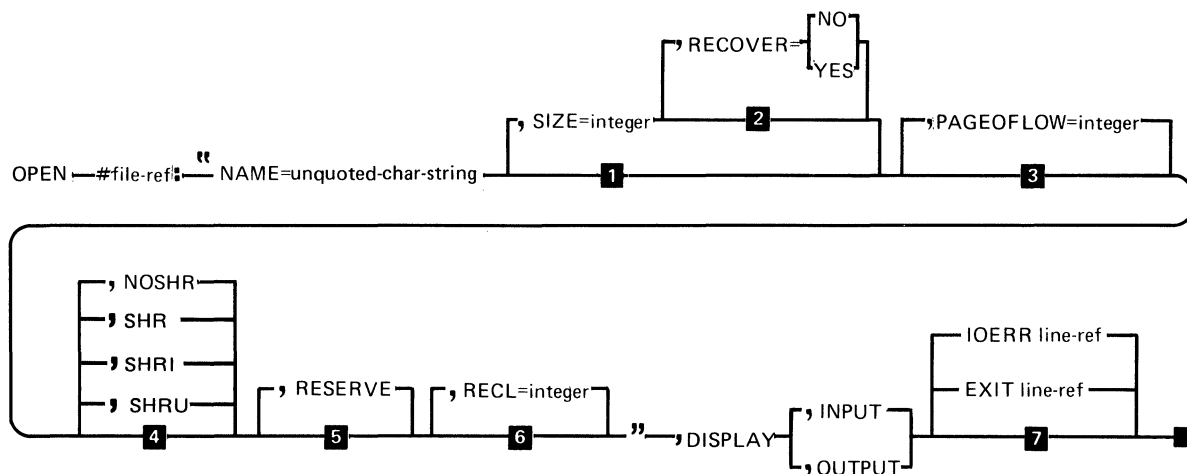
- Identify the file specification
- Assign a logical file reference
- Allocate initial space for new files
- Specify file usage
- Specify file type (**DISPLAY** or **INTERNAL**)

An **OPEN** statement must be issued for a file before any input or output statements access that file.

OPEN

OPEN statement (continued)

OPEN DISPLAY (syntax)



- 1** Old file
- 2** No backup label
- 3** Page length = 60 lines
- 4** No sharing allowed
- 5** Do not change old reserve status
- 6** 132
- 7** Interrupt on error unless ON ERROR is active

See "OPEN parameter table."

No blanks are allowed between a keyword and the equal (=) sign. The information in quotes is not syntax checked until the OPEN statement is executed. A character expression can be used in place of the information in quotes.

The syntax for OPEN DISPLAY is as shown, where:

file-ref is a numeric expression, (see "File reference parameter")

NAME= specifies the file specification for the file to be opened. The form and content of the file specification is described under "File specification parameter."

SIZE= specifies the size, in bytes, of a new file to be created. The *SIZE=* parameter must be specified for a new file and must not be used for existing files. The value is rounded up to the next higher multiple of 512 and the space is permanently allocated to the file. If the specified file already exists, an error will occur. The default (*SIZE=0*) indicates 4096 bytes. *DISPLAY* files are dynamically extended. The largest size you may specify is 16,777,215 bytes.

RECOVER= YES specifies that an extra copy of the file label should be created and maintained. In case an I/O error occurs for this file, the extra label makes it easier to recover data (see Recover Diskette under "Customer Support Functions") This keyword is ignored for files on the 5247 Disk.

RECOVER = NO specifies that an extra copy of the file label should not be created.

OPEN

OPEN statement (continued)

PAGEOFLOW= specifies the line number which when printed or exceeded, a *PAGEOFLOW* condition will exist. When the line being printed is \geq the *PAGEOFLOW* value a page overflow condition exists. The program may trap the *PAGEOFLOW* condition by specifying a *PAGEOFLOW* exit on a *PRINT* statement. Otherwise it is ignored. Transfer of control to a *PAGEOFLOW* exit occurs after all lines have been printed. A *PRINT #file-ref: NEWPAGE;;* must be executed to prevent another overflow condition on the next *PRINT* and to reset the *PAGEOFLOW* counter to 0 (range 1-255). Default is 60.

SHR specifies that any supported level of file sharing with another *OPEN* statement is permitted. Two opens for *OUTPUT* or *OUTIN* are not supported.

NOSHR specifies that no sharing is permitted.

SHRI specifies that this *OPEN* statement permits sharing files with other *OPEN* statements for input only.

SHRU specifies that any supported level of file sharing with another *OPEN* statement is permitted. Multiple *OPEN* statements are permitted for *OUTPUT* or *OUTIN*. The *SHRU* option applies only to files on the 5247 Disk.

RESERVE specifies that the *OPEN* sharing status is to be permanently associated with this file and station. The *RESERVE* option will be reset when a *CLOSE* statement specifying the *RELEASE* option is executed for this file. The *RESERVE* can also be reset using the *PROTECT* command.

RECL= Specifies the number of columns per physical line of PRINT output, formatted or unformatted. Range: 1–255.

DISPLAY specifies that the file to be opened is either the printer or a type 05 file. Do not specify SHRU/OUTPUT when you open a display file on the 5247 Disk.

INPUT specifies that data will be transmitted from the device using the INPUT or LINPUT statements.

OUTPUT specifies that data will be transmitted to the device using the PRINT statement.

EXIT specifies the line number or label of an EXIT statement to refer to if an I/O error occurs (see "EXIT statement").

IOERR specifies the line number or label to receive control if an error condition prevents the completion of the OPEN statement.

OPEN

OPEN statement (continued)

Example

```
100 OPEN #101:"NAME=PROC5/START,SIZE=512",  
    DISPLAY,OUTPUT
```

The following example can be used to open a file whether it exists or not:

```
•  
•  
•  
100 OPEN #1:"NAME=FILE/VOL,SIZE=1000",DISPLAY,  
    OUTPUT IOERR EXISTS ! Assume new file  
110 GOTO 150  
120 EXISTS: IF ERR<>4150 THEN STOP  
130 OPEN #1:"NAME=FILE/VOL",DISPLAY,OUTPUT  
    ! Old file  
140 RESTORE #1:  
150 . . .
```

To open to the system printer, use device address 10:

```
110 OPEN #55:"NAME=//10,RECL=255",DISPLAY,OUTPUT
```

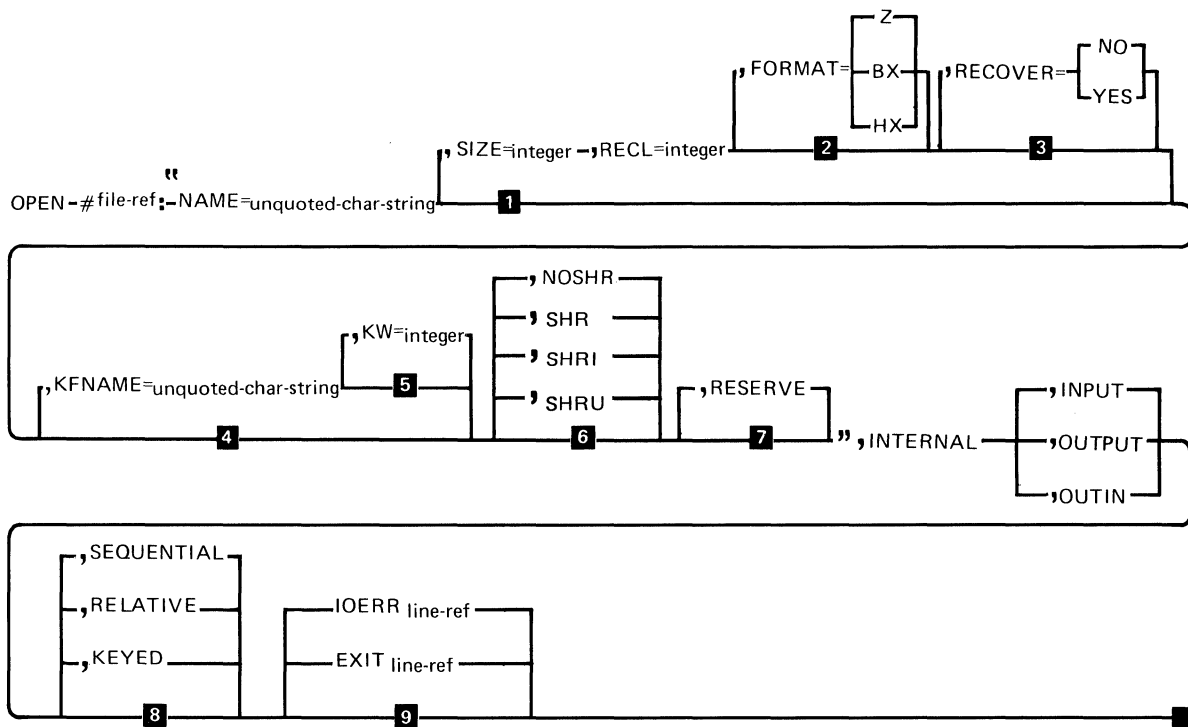
To open to the feature printer (second printer), use device address 11:

```
120 OPEN #56:"NAME=//11,RECL=192",DISPLAY,OUTPUT
```

This is the only way to access the feature printer.

See "Device address parameter" and "File specification parameter."

OPEN INTERNAL syntax



- 1** Old file
- 2** Format = Z (System/23)
- 3** No backup label
- 4** Not key accessed

OPEN

OPEN statement (continued)

- 5 Key work area = 0
- 6 No sharing allowed
- 7 Do not change old Reserve status
- 8 Sequential
- 9 Interrupt on error unless ON is active

See "OPEN parameter table."

No blanks are allowed between a keyword and the equal (=) sign. The information in quotes is not syntax checked until the OPEN statement is executed. A character expression can be used in place of the information in quotes.

The syntax for OPEN INTERNAL is as shown, where:

file-ref is a numeric expression, see "File reference parameter."

NAME= specifies the file specification for the file to be opened. The form and content of the file specification is described under "File specification parameter."

SIZE= specifies the size, in bytes, of a new file to be created. The *SIZE=* parameter must be specified for a new file and must not be used for existing files. *Size=* is in multiples of 512 bytes. For example:

Size=1 through 512 specifies 512 bytes. *Size=513* through 1024 specifies 1024 bytes. *Size=1025* through 1536 specifies 1536 bytes. *Size=1537* through 2048 specifies 2048 bytes, etc. The maximum size you can specify is 16,777,215 bytes.

If the specified file already exists, an error occurs. The default (*SIZE=0*) is 4096 bytes. *FORMAT=Z* files are dynamically extended.

FORMAT= BX (Basic exchange file), HX (H Exchange file), Z (System/23). Z is the default. BX and HX are invalid for files on the 5247 Disk. Type Z is the only format supported for disk files.

RECL= specifies the record length of the file being created. RECL must be specified for a new file and cannot be specified for an existing file. The maximum values which can be specified for RECL are 128, 256, and 4096 for *FORMAT=*BX, HX, and Z. For performance considerations, the following record lengths are recommended for *FORMAT=*Z files: 15, 31, 63, 127, 255, 511, 1023, 2047, and 4095. An example of an OPEN statement using *SIZE=* and *RECL=* is:

```
30 OPEN #1:"NAME=CUST//1,SIZE=0,RECL=127",  
INTERNAL,OUTPUT
```

For the Communications feature, the RECL parameter must be specified and represents the maximum record length for all data files transmitted or received throughout the communications session.

Maximum allowable record lengths are:

- Asynchronous communications, RECL=512
- Binary synchronous communications, RECL=128

RECOVER= YES specifies that an extra copy of the Data Set Label should be created and maintained. This keyword is ignored for files on the 5247 Disk.

RECOVER = NO specifies that an extra copy of the Data Set Label should not be created. RECOVER = NO is the default.

OPEN

OPEN statement (continued)

KFNAME= specifies the index file that is used to access the master file. This parameter is required when *KEYED* is specified. *KFNAME* is a file specification. See "File specification parameter."

KW= specifies the amount of work area to be used for accessing key indexed files. If *KW* is not specified or assigned the value of zero, no work area will be allocated. For optimum performance, see "Programming considerations" at the end of this *OPEN* statement.

SHR specifies that any supported level of file sharing with another *OPEN* statement is permitted. Two opens for *OUTPUT* or *OUTIN* are not permitted (see "File sharing").

NOSHR specifies that no sharing is permitted.

SHRI specifies that this *OPEN* statement permits sharing files with other *OPEN* statements for input only (see "File sharing").

SHRU specifies that any supported level of file sharing with another *OPEN* statement is permitted. Multiple *OPEN* statements are permitted for *OUTPUT* or *OUTIN*. The *SHRU* option applies only to files on the 5247 Disk.

RESERVE specifies that the *OPEN* sharing status is to be permanently associated with this file and station. The *RESERVE* will be reset when a *CLOSE* statement specifying the *RELEASE* option is executed for this file.

INTERNAL specifies that the file to be opened is a type 04 file (see "Internal I/O files").

INPUT specifies that input operations are performed on the file.

OUTPUT specifies that output operations are performed on the file.

OUTIN specifies that both input and output (update) operations are performed on the file.

SEQUENTIAL specifies that the file being opened is to be organized sequentially or accessed consecutively. *SEQUENTIAL* is the default.

RELATIVE specifies that the file being opened is a relative data set. Access to the file is random and is by record number.

KEYED specifies that the file being opened is a key-indexed file. Access is made through reference to user-defined keys which physically exist within each record in that file, and in an auxiliary key file.

EXIT specifies the line number or label of an EXIT statement to refer to if an I/O error occurs (see "EXIT statement").

IOERR specifies the line number or label to receive control if an error condition prevents the completion of the OPEN statement.

OPEN

OPEN statement (continued)

Example

This example assumes that N\$ and KFNS\$ contain the file specifications for the master file and key file:

```
10 OPEN #3:"NAME=" &N$ &" ,KFNAME=" &KFNS$ ,  
INTERNAL,OUTIN,KEYED
```

Programming considerations

- Positioning
 - INPUT and OUTIN files are positioned to the beginning. OUTPUT files are positioned to the end.
- BX and HX files
 - On an OPEN INTERNAL statement, if FORMAT=BX or FORMAT=HX is specified, file organization must be specified or defaulted to SEQUENTIAL: RECOVER=YES must not be specified; automatic file extension is not supported.

- OPEN #0 and #255
 - OPEN #0 is ignored.
 - OPEN #255 statement may be executed to override the default value for RECL and PAGEOFLOW.
- Extents
 - If additional space is required, up to 99 extents are created. Each extent is SIZE divided by 10, rounded up to the next 512 byte multiple.
 - System/23 Format files (Z) will be dynamically extended if necessary.
 - When a 5247 Disk file is extended and the new extent falls adjacent to the last extent, the two extents are merged.
- OPEN DISPLAY to the printer
 - If an OPEN DISPLAY is issued for the printer device, the file name or the VOLID in the file specification parameter will be ignored. The SIZE= and RECOVER= parameters cannot be specified.
- Record length
 - The record length of an internal file cannot be changed without first freeing the file and then recreating it.
- Device definition
 - If a new file is created (SIZE= parameter is specified) either VOLID or device address must be specified in the NAME= parameter.
 - If an old file is opened (SIZE= parameter is not specified) the VOLID and device address are optional.
- Implicit OPEN

OPEN

OPEN statement (continued)

- The only implied OPEN statements are for the system printer (file reference 255), and the system keyboard/display (file reference 0).
- File searches
 - If an old file is opened and no VOLID or device number is specified, and there is more than one file by that name, the file residing on the lowest numbered drive is opened (see "File searches").
- Key work area size

KW = Key work area size

KL = Key length

KS = Keys per block

NT = Number of tracks

INT = Integer system function

CEIL = Ceiling system function

ST = Number of blocks per track

NR = Number of records (Master File)

The minimum useful work area for an index file is $4+KL$. To obtain the maximum useful work area, specify $KW=65535$. The system will allocate only as much as it needs. The amount allocated by the system, when $KW=65535$ is specified, can be calculated as follows:

To find KS

$$KS = \text{INT}(512 / (4 + KL)) - 1$$

To find NT

$$NT = \text{CEIL}(NR / (KS * ST))$$

ST = 8 for type 1 and 2 diskettes 15 for type 2D diskettes and disk files

To find KW

$$KW = 2 + (NT * (KL + 2))$$

Example:

KL = 10, NR = 10000, 2D diskette

KS = INT(512/14) - 1 = 35 keys per sector

NT = CEIL(10000/(35*15)) = 20 tracks

KW = 2 + (20*(10+2)) = 242 bytes

The upper bound for KW is:

4736 for type 2D and 2 diskettes and disk files

18848 for type 1 diskette

See "Appendix A. Sample programs."

OPEN

OPEN statement (continued)

OPEN parameter table

Name	Open internal		Open display		
	Diskette or disk file		Diskette or disk file		Printer
	(new)	(old)	(new)	(old)	
filename	R	R	R	R	I
valid	O (1,5)	O (5)	O (1,5)	O (5)	I (5)
device	O (1)	O	O (1)	O	R
Size	R	(6)	R	(6)	E
Format	O	E	E	E	E
Recl	R	E	O	O (11)	O
Recover	O (2)	E	O	E	E
Kfname	E	O (3)	E	E	E
Kw	E	O (4)	E	E	E
Noshr	O (12)	O (12)	O	O	I
Shr	O (12)	O (12)	O	O	I
Shri	O (12)	O (12)	O	O	I
Shru (13)	O	O	O (14)	O	I
Reserve	O (12)	O (12)	O	O	I
Pageoflow	E	E	O	O (11)	O
Internal/display	R	R	R	R	R
Input/output/outin	R (9)	R	R (8,9)	R (8)	O
Sequential/relative/keyed	O (7,10)	O (7)	E	E	E
Exit	O	O	O	O	O
Ioerr	O	O	O	O	O

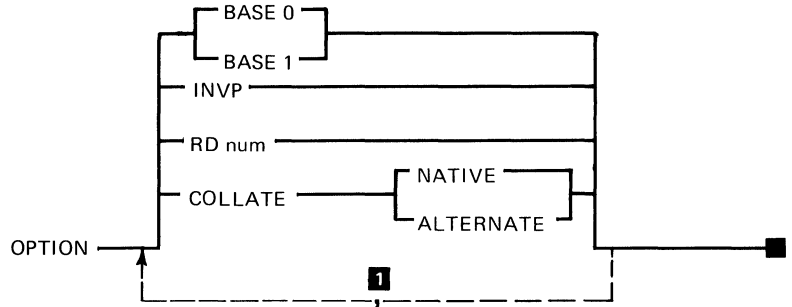
I = ignored R = required O = optional E = error

Notes:

1. Either volume or device is required.
2. Applicable to FORMAT=Z only. For FORMAT=BX or HX. RECOVER=YES is an error. RECOVER=NO is ignored. This keyword ignored for 5247 Disk files.
3. Required when KEYED specified.
4. Optional when KFNAME specified.
5. Specifying valid without filename is an error.
6. SIZE denotes a new file. SIZE must not be specified for an old file.
7. Specifying RELATIVE or KEYED for a FORMAT=BX or HX file is an error.
8. OUTIN is invalid on OPEN DISPLAY.
9. INPUT is invalid for new files.
10. KEYED is invalid for new files.
11. Ignored if file open for input.
12. Ignored for a FORMAT=BX or HX file.
13. SHRU valid for 5247 Disk files only.
14. Optional for Input only. Output will produce error.

OPTION statement

The OPTION statement specifies a set of options to be applied to the entire BASIC program. The OPTION statement may be placed anywhere in the program.



- 1** **BASE 0**
 American format
 Print all significant digits
 Native

The syntax of the OPTION statement is as shown above, where:

BASE 0 specifies that there is a zero row/column to any matrix.

BASE 1 specifies that the matrix starts with 1 in the row/column.

For more information, see "Arrays."

INVP (inverted print) specifies that the decimal point and the comma are interchanged in the printing of numeric values (European format). This interchange of decimal point and

OPTION

OPTION statement (continued)

comma affects the output from a PRINT or PRINT USING statement.

An example using PIC without INVP is: 123,456.78

The output with INVP is: 123.456,78

INVP also affects input using the N FORM statement data conversion specification. A comma in the input field will be treated as a decimal point.

RD num specifies how many rounded decimal digits are to be displayed to the right of the decimal point when a PRINT statement is executed. *nn* is an integer in the range 0 through 15.

If RD is not specified non-significant zeros will not be printed.

Note: RD affects only the printing of numbers. Internally the numbers do not change.

If RD 03 had been specified, 4.5678 would print as 4.568.

COLLATE NATIVE specifies the use of the system collating sequence, see "Character set."

COLLATE ALTERNATE specifies the use of the user specified sequence, see "Customer Support Functions" (change collating sequence,) "IF, THEN, ELSE statement," and "AIDX and DIDX."

Programming considerations

- CHAIN
 - When chaining is performed, the options of the chained-to program must be the same as the chained-from program, see "CHAIN statement."

Order of execution

The order of execution of BASIC statements in System/23 is given by the following rules:

After RUN verifies the global characteristics of the program (for example conflicting DIMs or OPTIONS), execution begins with the lowest numbered executable line not in a defined function.

The next line to be executed is the next higher numbered executable line, unless:

- A GOTO, GOSUB, Function reference, RETURN, NEXT, IF, RETRY, or CONTINUE specifies the next line.
- END, STOP or CHAIN terminates the program.
- PAUSE interrupts the program.
- DEF transfers control to the first executable statement following the function.
- If a computational error is detected (SOFLOW, CONV, OFLOW, UFLOW, ZDIV) for which an ON. .GOTO is active, the transfer takes place.
- If an I/O error is detected for which an EXIT is specified, transfer to the specified line occurs.
- If an I/O error, a computational error or INQ key is pressed and no applicable EXIT or ON condition GOTO is

Order of execution

Order of execution (continued)

specified, or the Cmd/Attn key is pressed, the program interrupts.

See "Interrupt handling" "ON statement" "Arithmetic hierarchy."

For a description of special handling of ON events and I/O exits within a defined function, see "DEF, FNEND statement."

ORD(A\$)

Returns the ordinal value of A\$, where A\$ has a length of 1 (the location in the collating sequence). This is affected by the Customer Support Function, Change Collating Sequence. See "OPTION statement."

If we use the native collating sequence:

A\$ ="0"

ORD(A\$) results in 240.

See Appendix B and "Character set."

Overstruck characters

See "FORM statement"

Packed decimal

See "FORM statement" (PD specification).

PAD

See "LPAD\$(C\$,X)" and "RPAD\$(A\$,X)".

Page overflow

See "OPEN statement"
"EXIT statement"
"PRINT statement."

Parameter, device address

See "Device address parameter."

Parameter, file reference

See "File reference parameter."

Parameter, file specification

See "File specification parameter."

PAUSE

PAUSE statement

The PAUSE statement interrupts program execution. It can be used for manually entering calculator operations or commands while execution of the program is suspended.

PAUSE 

When a PAUSE statement is encountered during program execution, execution is interrupted and:

PAUSE

is shown on the status line with the line number of the PAUSE statement. To resume program operation, issue a GO command (see "GO command"). Statements must not be renumbered or a calling statement altered while the program is interrupted.

When PAUSE is encountered, the screen will be split and the bottom 5 lines will not appear. If the required function is simply to stop execution, LINPUT can be issued instead (see "Split screens," "LINPUT statement"). A PAUSE may not be included within a function that you define.

PI

See "Internal constants."

PIC\$(C\$)

Returns the current currency symbol. **C\$**, which sets the new currency symbol, is optional. **C\$** must be one character in length. If **PIC\$** is not set by the program, it will be "\$." This setting is changed when power is switched on or by another **PIC\$** setting.

POS(A\$,B\$,X)

Returns the value of the first character of a substring in **A\$** that matches **B\$**.

X contains the character position in **A\$**, where the search for **B\$** is to start. If the substring indicated by **B\$** does not occur in **A\$**, zero is returned. For example:

```
10 A$="ABBCABCDE"  
20 B$="BC"  
30 P=POS(A$,B$,4)
```

P contains 6.

PRINT BELL

See "PRINT statement."

Print control

See "Printer assignment" and "PRINT statement."

Print data list delimiters

Print data list delimiters

See "PRINT statement."

Printer assignment

When a printer is assigned, an association is made between the printer and a file reference number. A printer can be assigned to only one file reference number at a time. The following table shows the BASIC statements and commands which assign printers.

Statement/command	File reference #	Device address (10 = system) (11 = feature)
PRINT executed in a BASIC program or in calculator mode. (see note)	#255	10
OPEN #n statement	file reference specified	device address specified
LISTP command	255	10
DIR n,PRINT	255	10
RUN TRACEP	255	10

Note: Except when the program was initiated by RUN DISPLAY.

A printer is released at the following times:

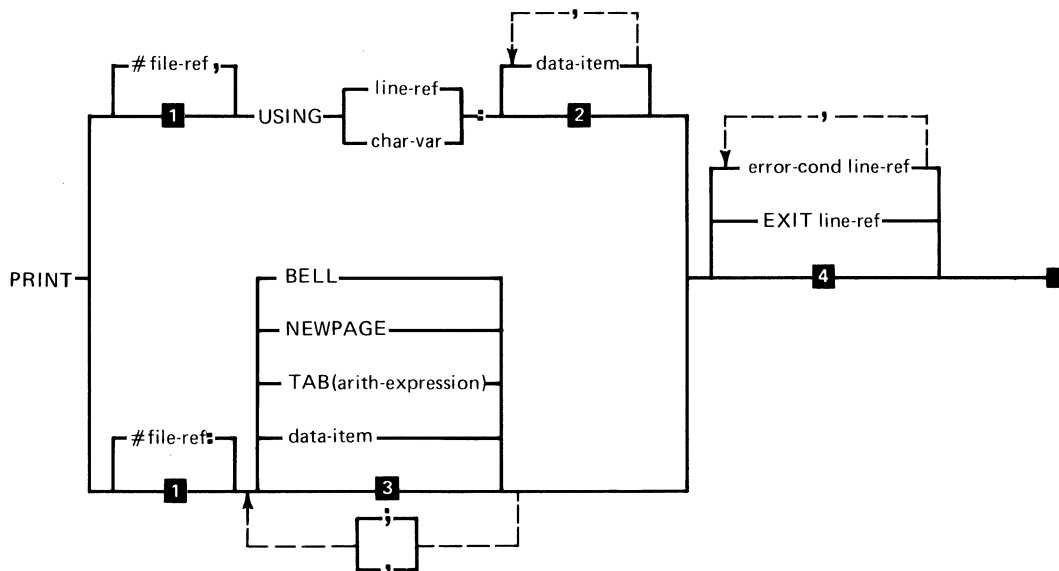
- When the RUN statement is entered (before BASIC program execution begins)
- When a CLOSE statement is executed to close a printer file
- When a BASIC program ends
- When a program is executed with RUN TRACEP, the system printer cannot be opened to a file reference number other than 255.
- If the operator enters a PRINT #255 statement, a LISTP, or a DIR n,PRINT command while a BASIC program is interrupted, the system printer is assigned to file reference number 255. This prevents the successful open of the printer to an alternate file reference number when the program resumes.
- Once the BASIC program assigns the system printer to an alternate file reference number, a PRINT #255 statement, LISTP, or DIR n,PRINT command entered by the operator while the BASIC program is interrupted will fail.
- In a BASIC program, when the system printer is assigned to file reference number 255, a CLOSE #255: statement will release it. It doesn't matter how the printer was originally assigned. If the system printer is not assigned, a CLOSE #255: is ignored.

PRINT

PRINT statement

The PRINT statement causes the values of specified scalar expressions or arrays to be displayed or printed. All output is DISPLAY data.

The output of the PRINT statement can be directed to the screen, file, system printer, or feature printer. In order to direct the output to the feature printer, file-ref must have been specified on an OPEN statement with a device-id of 11.



- 1** #0
- 2** Use FORM statement
- 3** Null data item
- 4** Interrupt on error unless ON is active

The syntax of the PRINT statement is as shown, where:

file-ref is a numeric expression, see ("File reference parameter").

USING must be either a line reference or a character variable. Line reference is the statement number or label of the FORM statement that defines how the data is to be formatted. Character variable is a character variable containing format information identical to that in a FORM statement.

data-item is one of the following:

- *MAT array-name* is the name of a one- or two-dimensional array. An example of how to print an entire array is:

```
50 PRINT #255: MAT ARRAYNAME
```

- "*char-expression*" is a character expression (see "Character expressions").
- *arith-expression* is an arithmetic expression (see "Expressions").

BELL makes the alarm ring for .25 seconds.

NEWPAGE causes printing to begin on the next form, or clears the screen (see "NEWPAGE function" in this section).

TAB allows the alignment of columns of data (see "TAB function" in this section).

, (comma) if *USING* is not specified, the comma causes individual items to be printed in pre-established horizontal zones called print zones. Each zone is 24 character positions (see "Print zones" in this section).

PRINT

PRINT statement (continued)

;
; semicolon as a delimiter, causes a null string (no extra blanks or spaces) to be printed between two groups of characters. (See "Print zones" in this section).

error-cond can be CONV, EOF, IOERR, PAGEOFLOW, SOFLOW (see "EXIT statement"). *line-ref* may be either a line number or a label symbol.

EXIT line-ref specifies the line number or label of an EXIT statement to refer to if an error occurs.

Programming considerations

- Printing arrays
 - When a PRINT statement is executed, each array element is converted to the output format and displayed. Each array is displayed in row order. For an unformatted PRINT each row begins at the start of a new line. Each array element is displayed or printed in succeeding print zones. Each element of a one-dimensional array begins on a new line.
 - For an unformatted PRINT, if an array is specified in the I/O list, no additional I/O list item or delimiter should follow.
- Formatted PRINT
 - When a PRINT USING statement is executed, the specified expressions or array references are evaluated. Their values are then edited into the corresponding format specifications in the specified FORM.

- Unformatted PRINT
 - Unformatted output consists of a PRINT statement with no USING clause. Spacing between displayed values is controlled by commas, semicolons, NEWPAGE, and TAB expressions.
 - When an unformatted PRINT statement is executed, the value of each specified expression is converted to the appropriate output format and displayed or printed in a left-to-right sequence, in the order in which it appears in the PRINT statement.
- Print zones
 - Each line that is printed or displayed is divided into print zones. Print zones are 24 character positions in length and are specified by the comma delimiter. For example:

```
200 PRINT A,B,C
```

This statement displays the value of the variable A, beginning in the first position of the new line. Since A is a positive value, column 1 is a blank character. The value of B will begin in position 25 of the same line and the value of C will begin in position 49 of the same line. If a character value is longer than 24 characters it uses as many zones as necessary to accommodate it. If a character or arithmetic value will not fit in the space remaining on the line, it will start on the next line.

PRINT

PRINT statement (continued)

```
160 A = 25.3
170 B = 66
180 C = -250
200 PRINT A, B, C
```

results in the following line being printed:

```
25.3      6.6      -250
↑         ↑         ↑
column2   column26  column 49
```

- Print data list delimiters
 - The meaning of comma, semicolon, and blank (b), when used in data lists is shown in the following chart.

	Trailing	Imbedded
,	Same line, next zone	Same line, next zone
;	Same line, next character	Same line, next character
b	New line	Error

- “Trailing” and “imbedded” refer to the location of the comma, the semicolon, and the blank in the PRINT statement data list.

- TAB function
 - The TAB (expression) function is used to align columns of data, in a manner similar to the TAB key on a typewriter. The TAB value must always be positive and if a non-integer, it is rounded. If TAB is negative, 1 is assumed. TAB(n) starts the next output in column n of the line. If the current position in the line is greater than n, data is put on the next line in position n.
- Printing data-items
 - *Character constants*:the actual characters enclosed in quotation marks are printed or displayed. In order to represent quotes, two consecutive quotes must be entered.
 - *Character variables*:the actual characters (excluding trailing blank) are printed or displayed.
- Numeric values
 - If OPTION RD is not specified and the value is between IE-6 and IE21, the number is printed in fixed format.
 - If OPTION RD is specified and the value is between IE-RD and (IE20-RD), the number is printed in fixed format. In all other cases, floating format is used.
- NEWPAGE function
 - On the printer, NEWPAGE causes printing to begin on the next page or form. PRINT #255:NEWPAGE;; sets the overflow line counter to zero.

On the display, NEWPAGE clears the screen and places the cursor in position 1 of line 23 (input line).

PRINT

PRINT statement (continued)

- 5241 and 5242 Printer statements –For printer spacing, line control, vertical or horizontal density, use the following statement:

```
PRINT #255: HEX$("2B020500hhvvlp")
```

- All numbers must be represented in hexadecimal form. The numbers in parentheses below are hexadecimal numbers.
- *hh* (horizontal density) is the number of characters per inch. The number of characters per inch can be 10 or 15 (Hex 0A or 0F) only. The default value set by the system is 10 characters per inch (Hex 0A).
- *vv* (vertical density) increments of 1/96 inch per line feed. The minimum number specified is 8 (Hex 08) and the maximum number is 99 (Hex 63). The default value set by the system is 6 lines per inch (Hex 10).
- *lp* (lines per page) lines per page can be 1–255 (1–FF). The default value set by the system is 66 lines per page (Hex 42).
- Specifying 00 for the above values, *hh*, *vv*, or *lp* does not change the current setting. If a value outside the allowable range is specified, the power-on default is used.
- To turn quality print on/off, use the following statement:

```
PRINT #255: HEX$("2BD10705FFxxyy0000") where
```

```
xx (font) = 00, economy mode (for drafts)  
           01, final mode
```

```
yy (type style) = 00
```

- The default printer control values can be reestablished by another PRINT #255 statement specifying the defaults. The defaults are also reestablished when the printer is powered on. Neither the CLOSE statement nor program termination will reestablish the defaults.
- To prevent a blank line from being output when printer control values are printed, specify SKIP 0 on a FORM statement referenced by the PRINT statement. For example,

```
10 PRINT #255,USING 20: HEX$("2B0205000A1042")
20 FORM C 7,SKIP 0
```

- Specify #255 for the system printer. The file reference number to be specified for a feature printer is the file reference number associated with that printer in the OPEN statement.
- During quality print mode, the horizontal density cannot be changed. See "Full-screen processing."

5217 Printer statements

Use the following statements if you are using the 5217 Printer. All the numbers in parentheses are in hexadecimal form and the values you input must also be in hexadecimal form.

To control the number of lines per inch, the number of characters per inch, and the number of lines per page use the following statement:

```
Print #255: HEX$("2B020500hhvvlp") where:
```

- *2B020500* remains unchanged
- *hh* lets you choose the number of characters per inch.

PRINT

PRINT statement (continued)

00= no change

0A= 10 characters per inch. This is the default.

0C= 12 characters per inch.

0F= 15 characters per inch.

- *vv* lets you choose the number of lines per inch in increments of 1/96 of an inch.

00= no change from the last setting.

10= 6 lines per inch. This is the default.

0C= 8 lines per inch.

12= 5 1/3 lines per inch. This is used for System/23 Word Processing.

Note: You may choose values from hex 01-63. Values larger than hex 63 will default to hex 10, or 6 lines per inch.

- */p* lets you choose the number of lines per page.

00= no change from the last setting.

3C= 60 lines per page

42= 66 lines per page. This is the default.

58= 88 lines per page

Note: Any other choice will result in the decimal value of the hexadecimal number.

Example:

```
Print #255: HEX$("2B0205000A1042")
```

This statement sets the printer to 10 characters per inch, 6 lines per inch, and 66 lines per page.

To control the ribbon mode, set proportional spacing, specify a particular character set, and signal the need to change the print wheel, use the following statement:

```
Print #255: HEX$ ("2BD10705FFxyycstd") where:
```

- *2BD10705FF* remains unchanged.
- *xx* Lets you choose the ribbon mode.

00= economy setting (for drafts). This is the default.
01= final mode.

Note: All other values result in economy setting. Refer to *Using your 5217 Printer* for printer switch settings to support this feature.

- *yy* lets you choose proportional spacing.

00= no change from the last setting.
04= proportional spacing.

Note: All other values reset proportional spacing and set the printer to 12 characters per inch.

- *cs* lets you choose a character set on a print wheel.

67= the default to character set 103.

- *td* is a number you supply to signal the need to change the print wheel. You determine what this number is. This number is displayed on the printer when a print wheel changed is required.

Example:

```
Print #255: HEX$ ("2BD10705FF01000187")
```


PRINT

PRINT statement (continued)

This statement sets the ribbon to final mode, with no change in the spacing from the last setting, asks for character set 001 and displays 87 to signal the need for a print wheel change.

To justify or leave standard the right margin, by adjusting the space size within the printed line, use the following statement:

Print #255: HEX\$("2BD2040Dstpc") where:

- *2BD2040D* remains unchanged.
- *st* lets you choose standard space size or to have the printer increase the space size between words. The amount of increase is controlled by *pc*.

00= Space size is standard (unjustified). This is the default.

01= Space size is increased (justified).

Note: Any other value will leave space size standard, or unjustified.

- *pc* lets you choose the way space size will be increased. This value is ignored if you have left space size standard.

00= no change from the last setting.

01-32= half justification, resulting in a ragged right margin.

33-64= full justification, resulting in an even right margin.

This is the default.

Note: Values 65-FF cause no change from the last setting.

Example:

Print #255: HEX\$("2BD2040D0164")

This statement specifies that space size will be increased causing the lines to end at the right margin.

To set the right margin, choose a paper drawer or choose to have a message returned if a statement exceeds the printers capability, use the following statement:

Print #255: HEX\$("2B020501rmscdi") where:

- *2B020501* remains unchanged.
- *rm* lets you set the right margin by specifying the number of characters per line.

00= no change from the last setting.

84= right margin is 132 characters from the left edge of the page. This is the maximum value you can input when printing at 10 characters per inch. Any value greater than 84 will default to 84. This is the default.

9E= right margin is 158 characters from the left edge of the page. This is the maximum number you can input when printing at 12 characters per inch. Values greater than 9E will default to 9E.

C6= right margin is 198 characters from the left edge of the page. This is the maximum value you can input when printing at 15 characters per inch.

Note: Any other choice will result in the decimal value of the hexadecimal number.

- *sc* lets you choose the drawer from which the paper is to feed.

00= no change from the last setting.

01= drawer 1 (bottom drawer). This is the default.

02= drawer 2 (top drawer).

03= drawer 1

04= drawer 2

PRINT

PRINT statement (continued)

Note: Any other value will cause drawer 1 to be used.

- *di* lets you make a choice when a coded statement requests something the printer cannot do. You may choose to have printing stop and a message returned, or to have printing continue.

00= no change from the last setting.

01= printing continues. This is the default.

02= printing stops and a message is returned.

Note: Any other value causes no change from the last setting. For a list of error conditions and the printer action when each condition is encountered, see *Using your 5217 Printer*, Appendix B.

Example

```
Print #255: HEX$( "2B020501840202" )
```

This statement prints a line 132 characters long, selects the paper from the top drawer and sends a message when a coded statement exceeds the printer's capability.

The following statements do not let you input values. They are used as they are, without change.

<i>Print #255 HEX\$("05 ")</i>	aligns data in columns for proportional spacing
<i>Print #255: HEX\$("16 ")</i>	backspaces the print head to the left one space.
<i>Print #255: HEX\$("1A ")</i>	backspaces the print head to the left 1/60 of an inch.
<i>Print #255: HEX\$("33 ")</i>	causes the printer to start a new line.

Print #255: HEX\$("0D ")


moves the print head to the start of the current line, without feeding a new line.

Print zones

See "PRINT statement."

PROC command

The PROC command initiates the use of a procedure file (see "Procedure file"). A procedure file is a DISPLAY (type 05) file on a diskette or on the disk, that can contain system commands, BASIC statements, and data. A procedure file allows you to set up the steps necessary to load and execute a series of BASIC programs (including data entry) without the need for operator intervention. For example, commands such as LOAD and RUN can be entered by executing the lines of a procedure file containing these commands. The lines of a procedure file are executed one line at a time, just as if they were entered from the keyboard.

PROC — file-spec — 

The syntax of the PROC command is as shown above, where:

file-spec consists of a filename followed by an optional volume identification and device address. A procedure file must be a type 05 file previously created and stored on a diskette or on the disk. For more information about file types, see

PROC

PROC command (continued)

“File types.” For the methods used to create this type of file, see “Procedure files.”

When the PROC command is executed, the file with the specified ID is accessed for procedure file data. The PROC command implicitly opens the procedure file.

See Programs 8 and 9-Sample in Appendix A.

Procedure file

The procedure file is a DISPLAY (type 05) file. A procedure file can contain commands, statements, and data to be used by the program for input.

Creating a procedure file

A procedure file can be created in two ways:

- From the keyboard using the CLEAR DATA command, followed by entering data lines containing commands. Data lines are preceded by a line number followed by a colon. After the file has been entered, it must be saved.
- Under program control using the OPEN DISPLAY statement. The file is written using PRINT statements, followed by a CLOSE statement.

Modifying a procedure file

As with any display file, a procedure file can be loaded as a data file and then modified or edited from the keyboard. The RENUM and AUTO commands can be used to allow entry of new lines.

Executing a procedure file

Use of a procedure file is initiated by a PROC or a SUBPROC command, which causes the file to be opened implicitly. Procedure files can also be called in a program by the CHAIN statement. Lines from the file are executed as if they were entered from the keyboard. (See "PROC command," "SUBPROC command," and "CHAIN statement.")

Procedure file

Procedure file (continued)

Closing a procedure file (after executing)

- The procedure file remains open while the procedure is active.
- It is closed by:
 - a PROC command embedded within the procedure that calls another procedure
 - issuing a CLEAR PROC command from the keyboard
 - an End Of File
 - GO END if no program is active
- A procedure can be exited by specifying the SKIP integer, where integer is a number larger than the number of lines remaining in the procedure.

A procedure can also be exited by issuing a GO END command from the keyboard when the procedure is interrupted.

Nesting a procedure file

Procedures may be nested by use of the SUBPROC command to a level of five active procedures or subprocedures. Procedures may provide input to Customer Support Functions.

SKIP

The SKIP command allows selective use of the lines in the procedure file (see "SKIP command").

ALERT

The ALERT command tells you that intervention is needed during execution of the procedure file (see "ALERT command").

Data

Data in a procedure file can also be used as input supplied in response to an INPUT or LINPUT statement (see "INPUT statement" "LINPUT statement" and "RUN command"). This is achieved by coding the PROC parameter in the RUN command.

Interrupting

If Cmd/Attn is pressed during execution of a procedure file, it is handled the same way as if an ALERT command had been encountered. Upon return to the procedure, the keyboard is opened before the next command in the procedure is executed. For more information, see "ALERT command" and "GO command." In both cases, enter GO to continue.

Procedure file

Procedure file (continued)

Storage

Each procedure file activated by a PROC or SUBPROC requires about 200 bytes of storage. The first requires 500 additional bytes.

Example

```
CLEAR DATA
10: LOAD PROG1
20: RUN
30: LOAD PROG2
40: RUN
50: ALERT INSERT PAYROLL DISKETTE THEN ENTER GO
  •
  •
  •
SAVE PAYROLL.PROC//2
```

In order to execute the procedure, the command PROC PAYROLL.PROC is entered.

PROG1 is loaded and executed; then, PROG2 is loaded and executed. The ALERT message is then displayed on the screen and the procedure stops.

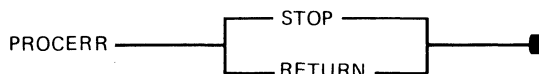
Programming consideration

- When a RUN command ends in an error (not as a result of an END or STOP command), be sure to enter CLEAR before you issue a LINK command.

For more information about PROC, refer to *Customer Support Functions*, Volume II under "Using a procedure file."

PROCERR command

The PROCERR command is a procedure file command that directs the system error handling mechanism either to return errors that occur in commands or are untrapped in BASIC programs to the procedure file or report them to the status line.



The syntax of the PROCERR command is as shown above, where:

STOP specifies that errors are to be reported to the status line. Pressing Error Reset, Cmd/Error Reset, or Cmd/Attn at this time opens the keyboard for input.

RETURN specifies that the next procedure file record is to be executed.

The PROC command sets the PROCERR option to STOP when issued with no procedure files active. Thereafter, the option may only be reset with the PROCERR command. The most recently issued PROCERR command from any level of procedure file nesting always controls the PROCERR option. The PROCERR command may be issued from the keyboard to alter the option when a procedure file has been interrupted (as by Cmd/Attn). This PROCERR will be considered the most recent until another PROCERR is issued.

Note: If the system release code field on your system status line is "1.05," the PROCERR command will be functionally different from preceding releases, and recoding of procedures may be required.

PROCERR

PROCERR command (continued)

If the system release code field on your system status line is "1.05," the PROC command will set the PROCERR option to STOP, whether issued from the keyboard or from a procedure file. Subprocedures will retain the PROCERR option of the invoking PROC unless explicitly coded otherwise.

The PROCERR RETURN command sets the value of the ERR variable to zero.

When the RETURN option is in effect, the next procedure file record is executed in command mode and the procedure continues. This should always be a SKIP command, which tests the value of the ERR variable, unless the error can be totally ignored (see examples).

Examples

PROCFILE.A

```
PROCERR RETURN !REGAIN CONTROL IF NO FILEXXX  
FREE FILEXXX/VOLXX  
PROCERR STOP !GOTO STATUS LINE IF LOAD/RUN FAILS  
LOAD FILEYY.PGM  
RUN
```

PROCFILE.B

```
PROCERR RETURN  
LINK COPY !COPY FILE1 TO FILE2  
.  
.  
ENDLINK  
SKIP 2 IF ERR=0 !IF COPY SUCCESSFUL CONTINUE  
ALERT COPY FAILED
```

Programming considerations

- The PROCERR RETURN option cannot be used to return to the procedure file when an OPTION message occurs. In the first example, the FREE command specifies a VOLID in the file specification to avoid an OPTION message in case the file does not exist.
- PROCERR RETURN will not trap syntax errors.
- PROCERR RETURN does not cause the procedure file to be searched for a valid command. If a program or Customer Support Function which reads data from procedure file fails to load, the next read from the procedure file will be data.
- If PROCERR RETURN is in effect during execution of a BASIC program and an untrapped error occurs, the BASIC program is only interrupted. It has not ended. Use caution if you want to have untrapped errors in BASIC programs return to procedure file execution.

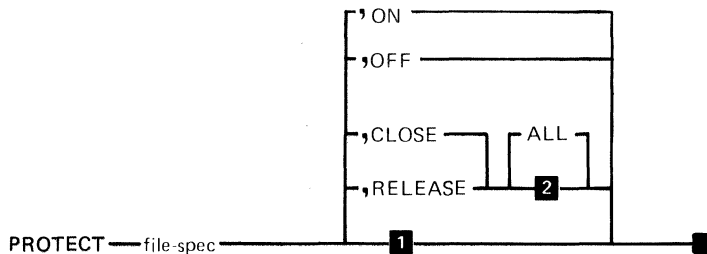
PROCIN

Indicates whether input is from the screen (0) or from a procedure file (1).

PROTECT

PROTECT command

The PROTECT command controls the integrity of data in a diskette or disk file by write-protecting the file. It can also be used to mark a file label as closed that has been left marked open or to RELEASE a file that has been reserved (see "DIR command" and "File sharing").



- 1** ON
- 2** CLOSE or RELEASE for this station only

The syntax of the PROTECT command is as shown above, where:

file-spec consists of a filename followed by an optional VOLID and device address. For more information, see "File specification parameter."

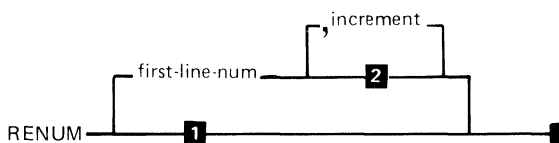
ON specifies that a file is to be write-protected.

OFF specifies that write protection is to be removed from a file that is currently write-protected. If none of the optional parameters are specified, ON is assumed.

CLOSE specifies that a diskette file label is to be updated to show that the file is not open for this station. For 5247 Disk files: CLOSE is ignored because your System/23 does it for you.

RENUM command

The RENUM command generates *new* line numbers for all BASIC program statements or data in the work area. Renumbering begins with line number 10; the increment is 10, unless otherwise specified. All references to line numbers such as in GOTO, IF, PRINT USING, GOSUB are changed to the new numbers.



1 RENUM 10,10

2 Increment is 10

The syntax of the RENUM command is as shown above, where:

first-line-num is the new beginning line number of the renumbered work area. If there is no *first-line-num* specified, a beginning number of 10 is the default value.

increment is the number specifying the increment for the succeeding statement numbers. If there is no increment number specified, an increment of 10 is the default value. If a *first-line-num* is not specified, increment cannot be specified.

RENUM

RENUM command (continued)

Example

This example shows the execution of a RENUM command:

```
RENUM 20, 15
```

Before

```
10 INPUT A,B&  
11 Q=INT (A/B)  
19 IF Q=-1 THEN STOP  
20 IF Q=0 THEN 30  
25 GOTO 10  
30 PRINT Q  
35 GOTO 10  
40 STOP
```

After

```
20 INPUT A,B  
35 Q=INT (A/B)  
50 IF Q= -1 THEN STOP  
65 IF Q=.0 THEN 95  
80 GOTO 20  
95 PRINT Q  
110 GOTO 20  
125 STOP
```

Programming considerations

- Interrupted program
 - The RENUM command is not valid during interrupted program execution.
- Permanent renumbering
 - After using RENUM, use the REPLACE command to update the file stored on the diskette or disk because RENUM only changes the file in the work area.

For related information, see "MERGE command."

REPLACE command

The REPLACE command saves the contents of the work area to an existing file. This command is similar to SAVE but applies only to files in which no name change is intended

REPLACE

REPLACE command (continued)

Programming considerations

- Diskette or disk full
 - If there is not enough space on the diskette or disk to save the entire file, an error occurs. In this case, the file should be saved on another diskette, or unneeded files should be freed to provide space.
- No file specification
 - SOURCE must be preceded by a comma if no file specification is specified; otherwise, SOURCE will be assumed for the filename.
- Cmd/Attn
 - Cmd/Attn will not interrupt during a REPLACE.
- Interrupted program
 - The REPLACE command cannot be issued if the program is interrupted; enter GO END before issuing REPLACE.
- Compressing the work area
 - No additional storage becomes available as a result of editing. Additional storage will be available if the program is saved or replaced in source format, and then loaded.
- Locked programs
 - The LOCK function replaces the program that is in memory in a format that prevents the source code from being listed. Since there is no UNLOCK function, the programmer should keep an unlocked version of the program in case the source code needs to be reviewed or modified.
 - While the LOCK function is provided as a safeguard, program security remains the responsibility of the author. It is not feasible to devise a LOCK that

- EOF—an attempt to read more data than provided by DATA statements
- SOFLOW—string overflow

See “EXIT statement” for more information on these error conditions.

EXIT line-ref specifies the line number or label of an EXIT statement that the system should reference if an error occurs.

At the beginning of program execution, a pointer is set to the first value in the internal data table specified by one or more DATA statements.

When a READ statement is encountered, successive values from the internal data table are assigned to variables and arrays in the READ statement beginning at the current data file position.

Example

```
CLEAR
10 OPTION BASE 1
20 DIM D(5)
30 DATA 5,10,15
40 READ A,B,C,MAT D
50 DATA 1,2,3,4,5
```

The values in the DATA statement are assigned in the same order to the variables listed in the READ statement. Once the READ and DATA statements are executed, A is equal to 5, B is equal to 10, C is equal to 15, and the five elements of array D are 1, 2, 3, 4, 5.

READ

READ statement (with no file reference) (continued)

Programming considerations

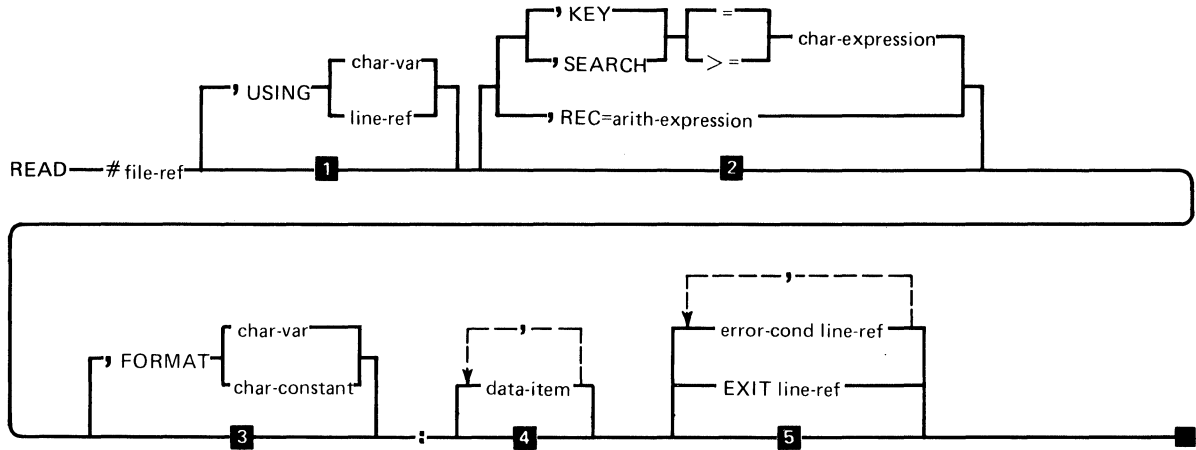
- Character assignments
 - The length of the character data item being read determines the length of the character string assigned to the character variable.
 - Numerical assignments
 - A string of digits not in quotes may be read as a number or a character string, depending on the variable type.
 - Array assignments
 - The array references in the data-item list are assigned values from the data file by rows, starting at the current data file position.
 - Truncation
 - If the numeric data exceeds 15 digits, truncation will occur.
 - Error conditions
 - A numeric data value was read and the READ statement specifies a character data item.
 - The data file is exhausted or no DATA statements exist in the program and unassigned data list items remain in the READ statement (EOF).
- If a READ statement is executed and there are no DATA statements in the program, an error will occur.
- The absolute value of a numeric data item is greater than the largest System/23 numeric value (see "Magnitude" under "Arithmetic data"). An overflow condition is generated.

- The absolute value of a numeric data item is less than the smallest System/23 numeric value (see "Magnitude" under "Arithmetic data"). An underflow condition is generated.
- The data that is read exceeds the length of the data item (SOFLOW).
- BASE
 - If the default (BASE 0) is in effect, an array dimensioned to size N requires N + 1 elements. See "DIM statement" and "OPTION statement."

READ

READ statement (with file reference)

The READ statement assigns values from records in a file to specified variables or arrays.



- 1 Unformatted READ
- 2 Read next sequential record
- 3 Communications feature not in use
- 4 Read record, do not use data
- 5 Interrupt on error unless ON is active

The syntax for the READ statement is as shown above, where:

file-ref is a numeric expression. See "File reference parameter."

USING specifies a *line-ref* (line reference) of a FORM statement or a *char-var* (character variable) containing a FORM statement. *Line-ref* can be a line number or label. The FORM statement is used to indicate the representation and location

to be assigned to the variables in the data-item list that will be read. Include a data-item list when you have a USING clause. If you do not have a USING clause, an unformatted read will be performed. See "Internal I/O file formatting."

KEY specifies the key field used to access the record in the file. The character expression must be the same length as the key field.

SEARCH specifies the key field used to access the record in the file. The character expression can be less than or equal to the length of the key field. If the character expression is shorter in length than the key field, the search of the index will consider only that part of the key field equal to the length of the specified character expression.

= specifies that the KEY/SEARCH argument must make an exact match to the record key.

>= specifies that if an equal compare is not made, the next record in the key sequence following the provided key is used.

REC=arith-expression is a positive, non-zero integer or numeric variable that specifies the logical record number of the record to be retrieved. If this parameter is not specified, the next logical record will be accessed.

data-item specifies the names of variables to be read into from the file. Data items can include elements, or entire arrays (preceded by MAT). The data items must be separated by a comma. The first data item must be preceded by a colon.

error-cond line-ref specifies the line number or label that the program should transfer to if one of the error conditions occurs. The following error conditions may be included in any order:

READ

READ statement (with file reference) (continued)

- CONV—conversion error
- EOF—end of file
- IOERR—input/output error
- NOKEY—key not found; invalid key reference
- NOREC—no record found; invalid record reference
- SOFLOW—string overflow

See "EXIT statement" for more information on these error conditions.

EXIT line-ref specifies the line number or label of an EXIT statement that the system should reference if an error occurs.

FORMAT is a communications feature clause. It specifies that special control functions are requested. The control functions can be specified as a *char-constant* (character constant) or in a *char-var* (character variable). See *System/23 Communications Guide*.

Example

```
10 OPEN #1:"NAME=ITEMS",INTERNAL,INPUT,RELATIVE
20 LET I=25
30 READ #1,USING 40, REC=I:I$,D$,A,B,C NOREC QUIT
40 FORM C 5,C 10,N 6,N 9.2,N 17.2
  •
  •
90 QUIT: CLOSE #1:
```

Record number 25 of the file ITEMS is read in statement 30.

An unformatted READ into an array that is too large will cause an error. The following example will read variable size, unformatted records containing numeric data.

```

10 OPTION BASE 1
20 DIM INARRAY (50)
  •
  •
80 READ #1,USING 90:L$,H$
90 FORM C 1,C 1
100 LET COUNT=ORD(L$)+256*ORD(H$)
110 MAT INARRAY=INARRAY(COUNT)
120 REREAD #1:MAT INARRAY

```

The following example uses KEYED:

```

10 OPEN #30:"NAME=KMAST,KFNAME=KINDX",
      INTERNAL,INPUT,KEYED
20 A$="NEAT"
30 READ #30,USING 40,KEY=A$:B$,F$,
      X NOKEY DONE
40 FORM C 4,X 5,C 16,X 2,N 3
  •
  •
  •
100 DONE: CLOSE #30:

```

The first record with a key field equal to NEAT is read in statement 30.

READ

READ statement (with file reference) (continued)

Programming considerations

- No positional specifications
 - If the file was opened with the KEYED parameter and if the KEY/SEARCH parameter is not entered, the next sequential record in the file is accessed in ascending key sequence. If the file was opened with the RELATIVE parameter and the REC parameter is not specified, then the next sequential record is accessed.
- Key length
 - The length of the KEY parameter must be equal to the record key field.
 - The length of the SEARCH parameter must be less than or equal to the length of the key.
- Skipping records
 - The data-item parameter can be omitted on READ to allow for error checking. There will be no transfer of data to variables, but a record will be read.
- Unformatted READ
 - To correctly interpret an unformatted record, the data types of the READ input list must match, element by element, with the data types in the WRITE statement that created the record. If the input list contains an arithmetic data item and the field length is not 9, a CONV error occurs. No other errors are detected. Character strings of length 9 can be interpreted as numeric values; numeric values can be interpreted as character strings of length 9.

- Errors
 - If REC= is specified and the record is deleted or is greater than the largest record number, a NOREC error will occur.
 - If the KEY and SEARCH parameters are specified for a file which has not been opened as an index file, an error will occur.

Record I/O files

See "Internal I/O files."

REC(N)

REC returns the record number, in file N, of the last record processed. A -1 is returned if the file is not open or if it is a display or keyed file. Zero is returned if no records have been processed.

Redimensioning

Redimensioning arrays

Numeric and character arrays can be redimensioned according to the following rules:

- Both one-dimensional and two-dimensional arrays can be redimensioned.
- The total number of elements in any array after redimensioning must not exceed the number originally specified when the array was dimensioned.
- The number of dimensions can be changed. A one-dimensional array can become a two-dimensional array and a two-dimensional array can become a one-dimensional array.
- The maximum value for a dimension is 9999 or is determined by the available storage.
- An array can be redimensioned in a MAT assignment (simple) statement or by using the ZER or CON functions (see "MAT assignment (simple)" and "ZER and CON").
- The new dimensions for the array can be specified with either a constant or an expression. The expression cannot contain subscripts.
- Redimensioning cannot occur on input or output data-item lists.
- The array is allocated storage when it is first referenced. After redimensioning, the unused storage of the array can be reused.
- The maximum length of each element in a character array cannot be changed.

Example

```
10 OPTION BASE 1
20 DIM BIG(50,50)
.
.
.
100 BIG(37,42)=12345.6 !SPACE FOR ARRAY "BIG"
.
.
.
300 MAT BIG=ZER(1,1) !MOST OF SPACE FREED UP
```

The preceding example shows how an array can be redimensioned in line 300. Line 300 takes the 50 X 50 array and makes it a 1 X 1. The data in BIG (37, 42) is lost.

Referencing, substrings

See "Substring referencing."

Relational expressions

See "IF, THEN, ELSE statement."

Relative record files

A relative record file is composed of a sequence of equal-size records. A record may be empty or may contain data. Each record has a number associated with it (a relative record number), starting with one and extending up to the number of records contained within the file. The relative record number is an index by which a record may be accessed for input or output. This number is not in the record. Access, using a relative record number, is made independent of the contents of

Relative record

Relative record files (continued)

any other record. Access to a relative record file may be either on a random basis or on a sequential basis. Relative record files can be accessed by relative record number and then accessed sequentially.

Record-oriented statements that access keyed files may specify either a KEY clause or a SEARCH clause. Record-oriented statements that access relative record files may specify a REC clause. A REC clause is mutually exclusive with either a KEY or a SEARCH clause. WRITE to a relative record file must contain a REC= clause referencing a deleted record only. The location following the last record in the file is considered a deleted record. The maximum number of records is 16,777,215.

A relative record file is accessed by:

- CLOSE
- DELETE
- OPEN (internal)
- READ
- REREAD
- RESTORE
- REWRITE
- WRITE

RELEASE statement

The RELEASE statement removes any record locks you have set on a file. This makes the records on the file available for use by the other computers. The RELEASE statement applies only to files on the 5247 Disk Unit. The RELEASE statement is unrelated to the RELEASE option of the CLOSE command and the PROTECT command.

RELEASE `_____` #file reference `█`

The syntax is shown above, where:

File reference is a numeric expression. See "File reference."

Example

```
10 READ #1:...
   .
   .
   .
90 RESERVE #1:
100 REWRITE #1:...
110 READ #1:...
120 REWRITE #1:...
130 RELEASE #1:
```

Programming considerations

- The RELEASE statement is designed to be used in combination with the RESERVE statement, as shown in the example. See "RESERVE statement."

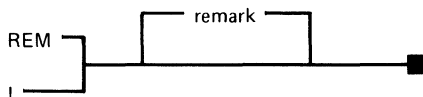
RELEASE

RELEASE statement (continued)

- The RELEASE statement may also be used to:
 - force data to be written to a disk file instead of being held in buffers.
 - remove locks that you implicitly set on a file using READ statements not followed by REWRITE statements.
- The FILE(N) function is not set by the RELEASE statement.
- A syntax error occurs when a variable named RELEASE is the target of an assignment statement that is not preceded by the word LET.

REM statement

The REM statement inserts remarks or comments in a BASIC program.



remark is one or more characters. This is an optional entry.

The REM statement is descriptive and is not executed. It appears in the program listing, but has no effect on program execution. "REM" uses 2 bytes of storage; "!" uses 4 bytes.

Example

```
10 REM THIS PROGRAM DETERMINES THE COST PER UNIT
or:
10 ! THIS PROGRAM DETERMINES THE COST PER UNIT
```

Remarks

Remarks may be added to a program by using the REM statement or an exclamation point, or to an existing statement by using an All blanks except one are deleted before and after the exclamation point. Lowercase characters are changed to uppercase unless enclosed in quotes. If blanks are required in the remark field, they must be preceded by a nonblank character (see example). Remarks are not permitted on DATA statements and should not be used on the DEL command.

Example

```
10 FOR MONTH=1 TO 12 !BEGIN LOOP
20 NEXT MONTH!"                end of loop"
30 A=B !*          SAVE B
```


RENAME

RENAME command

The RENAME command renames a file.

```
RENAME old-file-spec , new-file-name 
```

The syntax of the RENAME command is as shown above, where:

old-file-spec is the current file specification. For more information, see "File specification parameter."

new-file-name is the new name to be assigned to the file. This name must not already exist on the volume.

The RENAME command is rejected if *control reserve* is set for the file, if the file is in *open* status, or if the disk volume or diskette is access-protected. See "DIR command" and "File sharing."

Note: Because there is the potential for each file on the disk to have more than one person using that file, be careful when you rename a file. Programs and procedures must be changed to reference the new name.

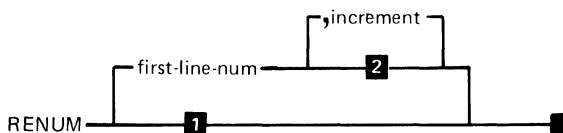
Example

```
RENAME OLD.FILE, NEW.FILE
```

This changes the name of the file OLD.FILE, to NEW.FILE.

RENUM command

The RENUM command generates *new* line numbers for all BASIC program statements or data in the work area. Renumbering begins with line number 10; the increment is 10, unless otherwise specified. All references to line numbers such as in GOTO, IF, PRINT USING, GOSUB are changed to the new numbers.



- 1 RENUM 10,10
- 2 Increment is 10

The syntax of the RENUM command is as shown above, where:

first-line-num is the new beginning line number of the renumbered work area. If there is no *first-line-num* specified, a beginning number of 10 is the default value.

increment is the number specifying the increment for the succeeding statement numbers. If there is no increment number specified, an increment of 10 is the default value. If a *first-line-num* is not specified, increment cannot be specified.

RENUM

RENUM command (continued)

Example

This example shows the execution of a RENUM command:

```
RENUM 20, 15
```

Before

```
10 INPUT A,B&
11 Q=INT (A/B)
19 IF Q=-1 THEN STOP
20 IF Q=0 THEN 30
25 GOTO 10
30 PRINT Q
35 GOTO 10
40 STOP
```

After

```
20 INPUT A,B
35 Q=INT (A/B)
50 IF Q= -1 THEN STOP
65 IF Q=.0 THEN 95
80 GOTO 20
95 PRINT Q
110 GOTO 20
125 STOP
```

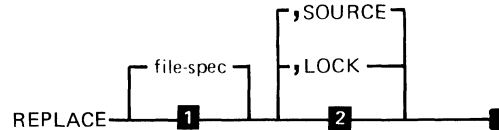
Programming considerations

- Interrupted program
 - The RENUM command is not valid during interrupted program execution.
- Permanent renumbering
 - After using RENUM, use the REPLACE command to update the file stored on the diskette or disk because RENUM only changes the file in the work area.

For related information, see "MERGE command."

REPLACE command

The REPLACE command saves the contents of the work area to an existing file. This command is similar to SAVE but applies only to files in which no name change is intended.



- 1** Use the filename last saved or loaded.
- 2** Internal format

The syntax of the REPLACE command is as shown above, where:

file-spec is the file specification. For more information, see "File specification parameter." If the file does not already exist, an error will occur.

SOURCE indicates that the program is to be saved in source format. If *SOURCE* is not specified, the program is saved in BASIC format. In either case, the file type must conform to the present type stored. *SOURCE* files are type 05; BASIC program files are type 09 or type 81.

LOCK indicates that the program is to be locked. A locked program may not be listed, saved, or replaced in source format. Once the program is locked, it cannot be unlocked. A copy of the unlocked program should be kept by the programmer.

REPLACE

REPLACE command (continued)

Programming considerations

- Diskette or disk full
 - If there is not enough space on the diskette or disk to save the entire file, an error occurs. In this case, the file should be saved on another diskette, or unneeded files should be freed to provide space.
- No file specification
 - SOURCE must be preceded by a comma if no file specification is specified; otherwise, SOURCE will be assumed for the filename.
- Cmd/Attn
 - Cmd/Attn will not interrupt during a REPLACE.
- Interrupted program
 - The REPLACE command cannot be issued if the program is interrupted; enter GO END before issuing REPLACE.
- Compressing the work area
 - No additional storage becomes available as a result of editing. Additional storage will be available if the program is saved or replaced in source format, and then loaded.

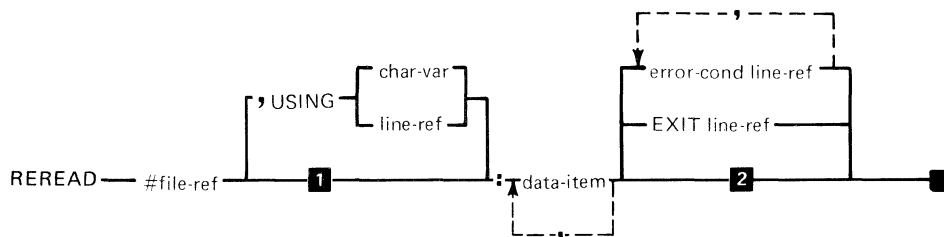
cannot be circumvented and IBM cannot take responsibility for the actions of those who do circumvent the LOCK function.

Replacing a statement

See "Editing a program."

REREAD statement

The REREAD statement assigns values from the most recent record READ or REREAD from the file.



- 1** Unformatted REREAD
- 2** Interrupt on error unless ON is active

The syntax for the REREAD statement is as shown above, where:

file-ref is a numeric expression. See "File reference parameter."

USING specifies a *line-ref* (line reference) of a FORM statement or a *char-var* (character variable) containing a FORM statement.

RERead

RERead statement (continued)

line-ref can be a line number or label. The FORM statement is used to indicate the representation and location of values to be assigned to variables in the input list that will be read. If no USING is specified, the record is read unformatted. See "Internal I/O file formatting."

data-item specifies the names of variables to be read into from the file. Data items can be variables, array elements, or entire arrays (preceded by MAT). The data items must be separated by a comma. The first data item must be preceded by a colon.

error-cond line-ref specifies the line number or label that the program should transfer to if one of the error conditions occurs. The following error conditions may be included in any order:

- CONV—conversion error
- IOERR—input/output error
- NOKEY—key not found; invalid key reference
- NOREC—no record found; invalid record reference
- SOFLOW—string overflow

See "EXIT statement" for more information on these error conditions.

EXIT line-ref specifies the line number or label of an EXIT statement that the system should reference if an error occurs.

Example

```
5 DIM D$*20
10 OPEN #1:"NAME=ITEMS",INTERNAL,INPUT,RELATIVE
20 FOR I=1 TO 100
30 READ #1,USING 40,REC=I:I$,D$
40 FORM C 5,V 20,N 6,N 9.2,N 17.2
50 IF D$="BOLTS" "BOLTS" THEN GOTO 80
60 REREAD #1,USING 40:I$,D$,Q,P,P1
70 NEXT I
80 ! CONTINUE OPERATION
```

RESERVE statement

The RESERVE statement is a request by the issuing computer for a temporary exclusive use of the file. RESERVE applies only to files opened on the 5247 Disk and is unrelated to the RESERVE option of the OPEN statement. When a RESERVE statement is coded, all other computers are prevented from accessing records in the file.

RESERVE — #file reference : —■

The syntax is shown above, where:

file reference is a numeric expression. See "File reference."

RESERVE

RESERVE statement (continued)

Exclusive use of a file is released when:

- a CLOSE or RELEASE statement specifying the same file reference is carried out.
- CLOSE is implied by a program ending; or when a GO END, CLEAR or CLEAR ALL command is issued.
- a PROTECT RELEASE command is issued. If a computer with a file locked through the RESERVE statement loses contact with the disk (because of power loss or system error, for example) the file remains locked until the PROTECT RELEASE command is issued.

Programming considerations

- A RESERVE statement may only be issued to a file opened with the same file reference.
- Use RESERVE to prevent other computers from accessing records while you are doing *multiple updates* to a file.
- *Single updates* are synchronized without the use of RESERVE.
- Locking the entire file also prevents access to records in the file that you are not using. An example of how to minimize this contention is:

```
10 READ #1:
   •
   •
   •
90 RESERVE #1:
100 REWRITE #1:...
110 READ #1:...
120 REWRITE #1:...
130 RELEASE #1:
```

- The READ at line 10 locks the record. Processing occurs with the single record locked. The file is locked by the RESERVE statement at line 90, so the exclusive use of the file begins at line 90. The RELEASE statement at line 130 removes the lock and ends the exclusive use status of the file.
- The RESERVE statement may also be used to lock out other computers while multiple reads are done.
- The FILE(N) function is not set by the RESERVE statement.
- An error message occurs if:
 - a computer has the same disk file open more than once under different file references and the file is already locked under one of the other file references.
 - RESERVE is the target of an assignment statement that is not preceded by the word LET.

Reserved words

Reserved words

The following words are reserved for System/23 BASIC and cannot be used as the name of variables or statement labels which you define.

ABS	EXP	NEWPAGE	RETRY
AIDX	FIELDS	NEXT	RETURN
AND	FILE	NOKEY	REWRITE
ATN	FILES	NONE	RLN
ATTN	FN	NOREC	RND
BELL	FNEND		
CEIL	FN---	OFLOW	ROUND
CHAIN	FOR	ON	SEQUENTIAL
CLOSE	FORM	OPEN	SGN
CMDKEY	FREESP	OPTION	SHIFT
CNT	GO	OR	SIN
CODE	GOSUB	ORD	SOFLOW
CON	GOTO	OUTIN	SQR
CONTINUE	IF	OUTPUT	SRCH
CONV	IGNORE	PAUSE	STEP
COS	INTERNAL	PI	STOP
DATA	INPUT		SUB
DATE	INT	PAGEFLOW	SYSTEM
DEF	IOERR	POS	TAB
DELETE	KEY	PRINT	TAN
DIDX	KEYED		THEN
DIM	KLN	PROCIN	TIME
DISPLAY	KPS	RANDOMIZE	TO
DISPLY	LEN	RD	TRACE
DUPREC	LET	READ	UDIM
ELSE	LINE	REC	UFLOW
END	LINPUT	RELATIVE	USE
EOF	LOG	REM	USING
ERR	MAT	REREAD	VAL
ERROR	MAX		WRITE
EXIT	MIN	RESTORE	ZDIV
			ZER


The following words cannot be used as the names of user-defined character variables.

CHR\$	KSTAT\$	RPAD\$	STR\$
DATE\$	LPAD\$	RPT\$	TIME\$
FILE\$	LTRM\$	RTRM\$	WSID\$
HEX\$	PIC\$	SREP\$	

RESTORE

RESTORE statement (with no file reference)

This RESTORE statement causes subsequent READ statements to assign values beginning with the first item in the first DATA statement (see "DATA statement").

RESTORE 

The syntax of the RESTORE statement is as shown above.

The RESTORE statement returns the internal data table pointer from its current position to the beginning of the table.

A RESTORE statement is ignored in a program that contains no DATA statements.

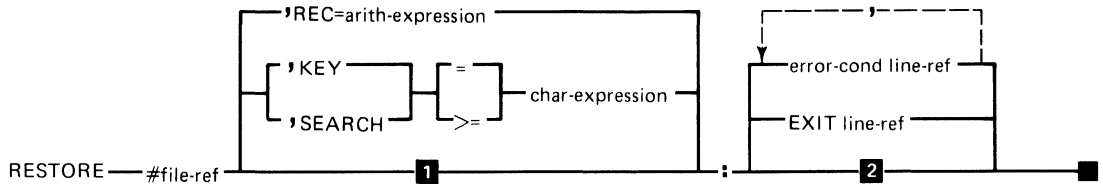
Example

```
10 DATA 1,2
20 READ A,B
30 RESTORE
40 READ C,D
```

In the above example, after the statements are executed, the variables A and C will each have a value of 1 and variables B and D will each have a value of 2.

RESTORE statement (with file reference)

This RESTORE statement is used to reposition a file.



- 1** Go to the beginning of the file (if opened for output, DROP the file)
- 2** Interrupt on error unless ON is active

The syntax of the RESTORE statement is as shown above, where:

file-ref is a numeric expression. See "File reference parameter."

When a RESTORE statement (without a parameter) is executed, the specified display or internal I/O file is repositioned so that subsequent references to the file will refer to the beginning of the file.

REC=arith-expression for relative access, specifies the number of the record to which the file will be reset. The record specified by the arithmetic expression will be the next record in the file to be accessed by a READ without a REC clause.

KEY for keyed access, specifies the key field used to access the record in the file. KEY indicates key-indexed access of the file, which must have been opened as a keyed file.

RESTORE

RESTORE statement (with file reference) (continued)

char-expression parameter contains the actual record key to be compared to those records in the file. The character expression must be the same length as the key field.

SEARCH for keyed access specifies the key field used to access the record in the file. *SEARCH* indicates key-indexed access of the file.

char-expression can be less than or equal to the length of the actual key field. If the *char-expression* is shorter than the key field, the search of the index will consider only that part of the key field equal to the length of the character expression.

= specifies that the *KEY/SEARCH* argument must make an exact match to the record key.

>= specifies that if an equal compare is not made, the next record in key sequence following the provided key is used.

error-cond line-ref specifies the line number or label that the program should transfer to for one of the error conditions. The following error conditions may be included in any order:

- *IOERR*—input/output error
- *NOKEY*—invalid key reference
- *NOREC*—invalid record reference

See "EXIT statement" for more information on these error conditions.

EXIT specifies the line number or label of an EXIT statement that the system should reference if an error occurs.

Programming considerations

- Adding data
 - To position at the end of data, close the file, then reopen the file for output.
- RESTORE #0
 - RESTORE with a file reference of #0 is ignored.
- Dropping data
 - A RESTORE statement that specifies no parameters, places the file at the beginning of data. For the following specific cases, previously valid data becomes inaccessible (same action as DROP command):
 1. A display file opened for output, NOSHR, sequential access.
 2. An internal file opened for output, NOSHR, sequential access.

An error occurs if a RESTORE statement is executed for an internal file opened for output with relative or keyed access.

RETRY

RETRY statement

The RETRY statement transfers control to the statement causing the most recent error that was not suppressed by an ON condition IGNORE. See "ON statement"

RETRY ————— ■

RETRY is useful following an ON GOTO transfer or following an I/O exit. Any event that can cause an ON GOTO transfer or an I/O exit will set the line to which RETRY will transfer control.

If an ON event is specified as IGNORED, the return statement specification used by RETRY is not changed.

Example

```
100 ON ZDIV GOTO FIX
.
.
.
300 INPUT C
310 Z = 10/C
.
.
.
400 FIX:   C= 10 ENTERED FOR C
410 RETRY ! REEXECUTE LINE 310
```


Programming considerations

- **RETRY without error**
 - If no error has occurred since RUN, execution of RETRY will cause an error and will interrupt execution.
- **RETRY after INQ**
 - Execution of RETRY following INQ causes the same line to be executed again.
- **Multiple errors**
 - If a second ON GOTO or I/O exit occurs before RETRY is executed, the first occurrence is lost. Avoid operations that can cause such occurrences or use ON IGNORE.

For a description of special handling of ON events and I/O exits within a defined function, see “DEF, FNEND statement.”

RETURN statement

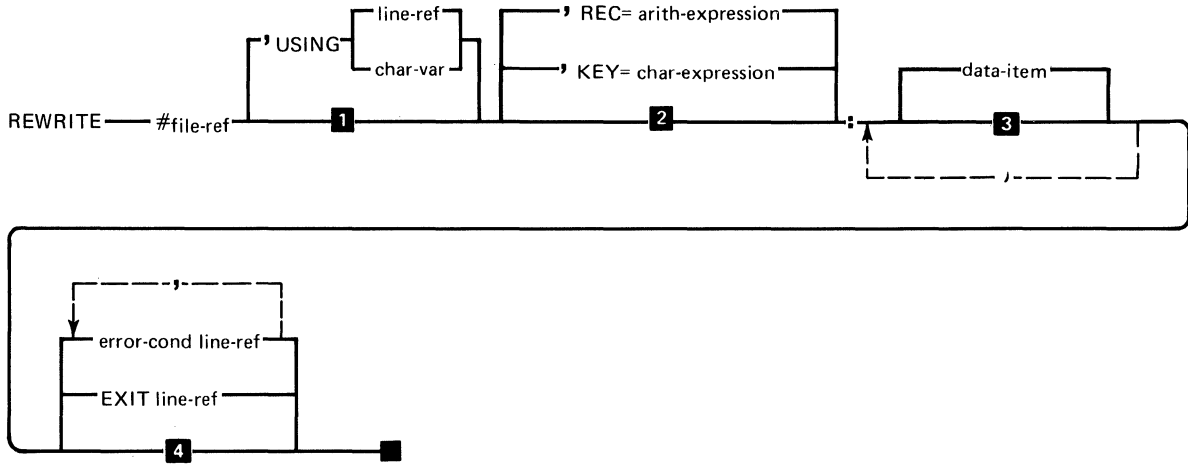
The RETURN statement transfers program control to the first executable statement following the most recently executed GOSUB statement (see “GOSUB statement”).

RETURN 

REWRITE

REWRITE statement

The REWRITE statement replaces an existing record in a file.



- 1 Unformatted write
- 2 Rewrite the last record READ/REREAD
- 3 REWRITE record with no data
- 4 Interrupt on errors unless ON is active

The syntax of the REWRITE statement is as shown above, where:

file-ref is a numeric expression. See "File reference parameter."

USING specifies a *line-ref* (line reference) of a FORM statement or a *char-var* (character variable) containing a FORM statement. *Line-ref* can be a line number or label. The FORM statement is used to indicate the type, length, and locations of the variable values (data items).

REC=arith-expression specifies the record having a relative record number equal to an arithmetic expression.

KEY=char-expression specifies the key field used to access the record in the file.

data-item specifies the names of variables or expressions that contain the values to be written to the file. Data items can include variables, array elements, entire arrays (preceded by MAT) or numeric or character expressions. The data items must be separated by a comma. The first data item must be preceded by a colon.

error-cond line-ref specifies the line number or label that the program should transfer to if one of the error conditions occurs. The following error conditions may be included in any order:

- CONV—conversion error
- EOF—end of file; insufficient file space for data
- IOERR—input/output error
- NOKEY—key not found; invalid key reference
- NOREC—no record found; invalid record reference
- SOFLOW—string overflow

See “EXIT statement” for more information on these error conditions.

EXIT line-ref specifies the line number or label of an EXIT statement that the program should transfer if an error occurs.

For more information, see Appendix B.

REWRITE

REWRITE statement (continued)

Example

```
10 OPEN #3: "NAME=FILEB", INTERNAL, OUTIN
20 DIM A$(6)*3
30 READ #3: MAT A$
40 A$(3) = "ABC"
50 A$(6) = "XYZ"
60 REWRITE #3: MAT A$
```

Programming considerations

- OPEN OUTIN
 - OUTIN must be specified in the OPEN INTERNAL statement.
- Preceding statements
 - REWRITE must be preceded by a successful READ or REREAD to the same file reference if no KEY or REC parameter is specified. If KEY or REC is specified, the record is read before it is updated and rewritten.
- No data
 - If the I/O list is omitted, there will be no transfer of data from variables. A record will be written.
- Key field
 - A REWRITE to a file opened for keyed processing cannot alter the key field.
 - A REWRITE may modify any field other than the one used for the associated key specified by OPEN. This includes fields used for other keys. If the field is modified, the associated key file must be regenerated by the INDEX Customer Support Function before it is used again. If this is not done, unpredictable modifications to the master file may result if

there is a subsequent WRITE/REWRITE/DELETE operation that uses that key field.

- There is no check made to verify that the Master file record obtained (using the index file) contains the key characters indicated by the KEY= specification.
- Communications
 - REWRITE is not applicable to Communications.
- Multiple rewrites
 - When a file is opened INTERNAL OUTIN with KEYED, or RELATIVE, a REWRITE statement without a record specifier (KEY= or REC=) can be used to update portions of a record that was just accessed by READ or REREAD. The record may not be processed by a second chronologically sequential REWRITE without another intervening READ (KEY= or REC=). See "I/O table 7" and "I/O table 8" in Appendix B.

If the record cannot be completely updated in one REWRITE because of a long FORM specification or data list, a second REWRITE may be required. This requirement can have the following effects:

- If the file is in SHR (shared) mode, the other program may access the updated record between the two (or more) REWRITE statements. This provides a copy of the record which is not valid to the READ statement in the other computer.
- If the record is one of a group with duplicate keys, the program may require considerable time to reaccess the record. Reaccessing the record takes awhile because the program must search through all the preceding duplicate keys.

REWRITE

REWRITE statement (continued)

These effects can be avoided by creating a temporary file with one record. This record should be the same size as the record in the data file. The following code will then update the record without excessive search time or loss of data integrity:

Assume a record length of 2000 bytes.

```
5 WORK=5
10 OPEN #WORK:"NAME=WORKFILE/VOLID,SIZE=2001,
    RECL=2000",INTERNAL,OUTIN,SEQUENTIAL
20 DIM A$(8)*250 ! USE THE MINIMUM NUMBER OF
    ELEMENTS MAXIMUM LENGTH
.
.
.
100 MAT A$=A$(8)
110 READ #DATAFILE,USING COPYFORM:MAT A$
120 COPYFORM: FORM C 250
130 RESTORE #WORK:
140 READ #WORK:
150 REWRITE #WORK,USING COPYFORM:MAT A$
160 MAT A$=A$(1)
.
. PERFORM MULTIPLE REWRITES TO WORK FILE
.
300 MAT A$=A$(8) ! REDIMENSION A$
310 RESTORE #WORK:
320 READ #WORK,USING COPYFORM:MAT A$
330 REWRITE #DATAFILE,USING COPYFORM:MAT A$
340 MAT A$=A$(1) ! RELEASE STORAGE
.
.
.
```

See "Program 5—Sample" in Appendix A.

RLN(N)

RLN returns the record length for internal file N. If file N is not open internal, a -1 is returned.

RND(X)

RND returns a random number in the range of 0 to 1. If X or the RANDOMIZE statement is specified, the random number generator is reset. Each random number is computed from the previous one according to a fixed algorithm. When X is specified, the number generated is the number that would normally follow X. The value specified for X must be greater than 0 and less than 1.

If X is not specified and RANDOMIZE is not executed, 2.1E9 unique numbers will be generated before the sequence repeats. "Run" starts the random numbers at the same point each time.

See "Program 2—Sample" in Appendix A.

ROUND(X,M)

ROUND(X,M)

ROUND returns the value of X rounded to M decimal digits to the right of the decimal point. If M is negative, X is rounded to the left of the decimal point (M trailing zeros following the number). For example:

```
10 X=15.735
20 R=ROUND (X,2)
```

R contains 15.74.

```
10 X=273
20 R=ROUND (X,-2)
```

R contains 300.

RPAD\$(A\$,X)

RPAD returns a string of characters of length X or greater by placing the required number of blanks to the right of A\$. If X is less than the length of A\$, then A\$ is returned unchanged. For example:

```
10 A$="ABCD"
20 B$=RPAD$(A$,5)
```

B\$ contains "ABCD ".

```
10 A$="ABCD"
20 B$=RPAD$(A$,2)
```

B\$ contains "ABCD."

Note: An error is generated if X is not within the 0 to 255 range.

RPT\$(A\$,M)

RPT\$ returns A\$ repeated M times. For example:

```
ABC$=RPT$("*",3)
```

ABC\$ Contains "****"

Note: When the result of RPT\$ exceeds 255 characters, the result will vary, based upon the function being performed.

RTRM\$(A\$)

RTRM\$ returns A\$ with all trailing blanks removed.

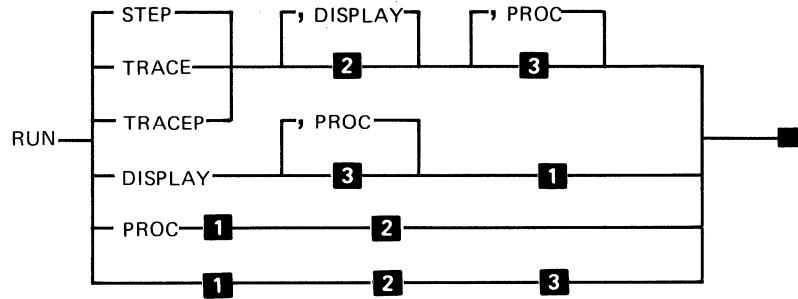
```
10 A$=" AB CD"  
20 B$=RTRM$(A$)
```

B contains "AB CD."

RUN

RUN command

The RUN command starts execution of a BASIC program at the lowest numbered executable statement. The program must reside in the work area, and the work area must be defined as containing a BASIC program.



- 1** Normal execution mode
- 2** Direct printed output to printer
- 3** INPUT, LINPUT #0 from keyboard

If DISPLAY or PROC parameter follows the RUN command, no comma is necessary.

The syntax of the RUN command is as shown, above where:

STEP specifies that the program stops before each statement is executed. The word STEP and the line number of the next statement to be executed are displayed. To execute the next statement, a GO command must be entered and you must press Enter. Execution will not stop inside functions you have defined.

TRACE specifies that the line number of each statement will be displayed when the statement is executed.

TRACEP specifies that the line number of each statement will be printed on the printer when the statement is executed. TRACEP should be used if tracing to the screen would overwrite valid information. The tracing information is accumulated until a line is full or until the printer is closed (for example, when the program terminates).

DISPLAY specifies that all PRINT #255 statements directed to the printer should be directed to the screen instead.

PROC specifies that data for INPUT and LINPUT statements should come from a procedure file rather than from the keyboard. This does not apply to INPUT FIELDS (see "Full-screen processing"). PROC is only valid on a RUN command issued from a procedure file.

Example

```
LOAD PAYROLL/LEDGER
RUN
```

A program named PAYROLL is loaded from a disk volume or diskette with VOLID LEDGER. The program is then executed.

Programming considerations

- Variable initialization
 - Each time the RUN command is issued, it initializes all arithmetic variables and arrays to zeros; all character variables and arrays are initialized to null.
- Resuming normal processing
 - If a RUN STEP is in process and normal processing is required, GO RUN can be issued.

RUN

RUN command (continued)

- Data work area
 - The RUN command will not be accepted by the system if the work area contains data rather than a program.
- Error conditions

The following errors will be detected before the first statement is executed:

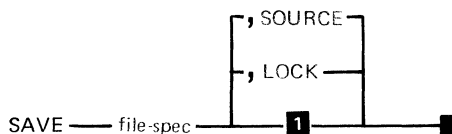
- An END statement appears and it is not the last statement.
- An undefined line number is found.
- FOR/NEXT loops are improperly nested.
- A previous RUN was interrupted. In this case, enter GO END, then RUN (see "Split screen").
- An array or character variable is dimensioned more than once.
- TRACEP and DISPLAY
 - If both TRACEP and DISPLAY are present, the trace information will be directed to the screen.
- TRACE
 - RUN TRACE(P) will trace the whole program. Use the TRACE statement to trace small portions of the program; see "TRACE statement."

Sample procedure or Sample program

See "Appendix A."

SAVE command

The SAVE command stores the contents of the work area in a specified file. SAVE is used to store a new program or data file for the first time, or an existing program or data file under a new name.



1 Save internal (file type 09 or 81)

The syntax of the SAVE command is as shown above, where:

file-spec is the file specification. For more information, see "File specification parameter."

Since SAVE stores programs or data files that do not already exist under the name you choose, the filename must be qualified by a VOLID or a drive number.

SOURCE indicates that the program is to be saved in source format as file type 05. If *SOURCE* is not specified, a program is saved in BASIC format as file type 09 or 81; data is saved as file type 05 (see "File types").

LOCK indicates that the program that is in memory is to be saved in a format that prevents the source code from being listed. Since there is no UNLOCK function, the programmer should keep an unlocked version of the program in case the source code needs to be reviewed or modified.

While the LOCK function is provided as a safeguard, program security remains the responsibility of the author. It is not feasible to devise a LOCK that cannot be circumvented and

SAVE

SAVE command (continued)

IBM cannot take responsibility for the actions of those who do circumvent the LOCK function.

Examples

```
SAVE MYPROG/MYVOL  
      or  
SAVE MYPROG.SRC//1, SOURCE
```

Programming considerations

- Locked programs
 - The LOCK function saves the program that is in memory in a format that prevents the source code from being listed. Since there is no UNLOCK function, the programmer should keep an unlocked version of the program in case the source code needs to be reviewed or modified.
 - While the LOCK function is provided as a safeguard, program security remains the responsibility of the author. It is not feasible to devise a LOCK that cannot be circumvented and IBM cannot take responsibility for the actions of those who do circumvent the LOCK function.
- Existing files
 - To save an existing program on a diskette or disk use REPLACE.
- Interrupted programs
 - SAVE cannot be issued if a program is in an interrupted state (from an error or Cmd/Attn, first enter GO END).

- Cmd/Attn
 - Cmd/Attn will not interrupt during a SAVE.
- Diskette full
 - If there is not enough space on the diskette to save the entire file, an error will occur. In this case, the file should be saved on another diskette.
- LOAD and SAVE
 - Program files (type 09 or 81) load and save faster than source files (type 05).
- Compressing the work area
 - No additional storage becomes available as a result of editing. The additional storage will be made available when the program is saved or replaced in SOURCE format, then loaded.

Scalar multiplication (MAT assignment)

See "Example (scalar multiplication)." under "MAT assignment (addition, subtraction, scalar multiplication)."

Search

See "SRCH(array,X[,row]) SRCH(array\$,X\$[,row])," or "File searches."

SGN(X)

SGN returns the sign of X.
SGN(-2) is -1 (representing a negative number).
SGN(+10) is +1 (representing a positive number).
SGN(0) is 0

Sharing

Sharing

See "Device sharing"
"File sharing"
"OPEN statement."

SHIFT(X)

SHIFT returns a value to indicate the machine type (Katakana or non-Katakana). The X parameter, which is optional, establishes a new shift mode.

Machine type	Value returned	Value of X
Non-Katakana	0	0 = lowercase 1 = uppercase
Katakana	1	0 = alphanumeric 1 = Katakana

The following PRINT statement will display the machine type:

```
10 PRINT SHIFT
```

The following LET statement will set a non-Katakana machine to uppercase shift mode:

```
20 LET A=SHIFT(1)
```

Sign of a number

See "SGN(X)" and "Arithmetic data."

Significance

See "Arithmetic data."

SIN(X)

SIN returns the sine of X. X is in radians and must be less than 1E10. Specify a value for X greater than $-2 * P1$ or less than $2 * P1$ for best accuracy.

Note: For correct results use:

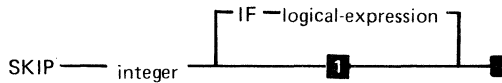
$SIN(X) = 2 * (\cos(\pi/4 - X/2))^{**2} - 1$.

(This note does not apply if the system release code field on your system status line is "1.05.")

SKIP

SKIP command

The SKIP command skips records within a procedure file.



1 Skip unconditionally

The syntax of the SKIP command is as shown, where:

integer indicates the number of records within a procedure file to be skipped.

logical-expression transfers procedure control according to the result of the logical expression (see "IF, THEN, ELSE statement"). Only CODE and ERR can be tested in the logical expression.

When the IF clause is specified on the SKIP command, the logical expression is evaluated. If the evaluation results in a true condition, the specified number of procedure file records are skipped. If the evaluation results in a false condition, no procedure file records are skipped.

The SKIP command without the IF clause causes the specified number of records to be skipped unconditionally.

Example

```
00010:LOAD FIRST
00020:RUN
00030:SKIP 3 IF CODE > 0
00040:LOAD SECOND
00050:RUN
00060:SKIP 3
00070:ALERT INSERT TRANSACTION DISKETTE
00080:LOAD THIRD
00090:RUN
```

In this example, the program in file FIRST is loaded and executed. If the program causes the value of the CODE variable to be set to positive, the SKIP 3 IF CODE > 0. causes the next three records in the procedure file to be skipped and the ALERT command to be processed. The program in file THIRD is then executed. If the value of the CODE variable from program FIRST is zero or less, the program in file SECOND is loaded and executed; then, "SKIP 3" unconditionally skips the last three commands.

The line numbers and colons are not part of the data on the procedure file. They are added and used by the editing commands.

See "PROCERR" and "Procedure files."

Skip Lines

See "FORM statement." (SKIP specification).

SORT

SORT command

The SORT command alters the order of the records in a file. SORT can be specified from a procedure file or entered from the keyboard.

SORT—*file-spec* —————■

The syntax of the SORT command is as shown above where:

file-spec is the file specification of a previously generated sort control file, which consists of a filename followed by an optional VOLID and device address (see "File specification parameter").

Information for the sort is always taken from the control file specified by the *file-spec*. See *Customer Support Functions*, Volume II for more information about SORT.

The SORT command is used in conjunction with the Sort Customer Support Function

Spaces

See "Blanks."

Special character set

See "Character set."

Split screen

A running BASIC program goes into split screen when the program is interrupted by:

- An untrapped error, followed by pressing Error Reset
- Pressing Cmd/Attn
- A PAUSE statement in the program

Lines 19 to 23 are temporarily replaced with a four-line blank area topped by a row of asterisks. This allows the entry of commands and desk calculator operations without disturbing the original screen.

If you enter any command which restarts or ends the program, the "split screen" is removed and replaced with the former display. Use the GO command to restart and GO END or CLEAR to end (see "GO command").

SQR(X)

SQR returns the square root of X. If X is less than zero, an error occurs.

Square roots

See "SQR(X)."

SRCH (array,X[,row]) or SRCH (array\$,X\$[,row,])

SRCH searches a one-dimensional array for the value of X. The result is a number that indicates in which row the argument X was found. "Row", which is optional, is used to select the starting row within the array. The default starting row is 0 (BASE 0) or 1 (BASE 1). If the argument is not found, -1 is returned. If the array has never had any data assigned to it, a -1 is returned.

Notes:

1. The value of "row", if supplied, must be within the range of 0 to 255. An error (405) occurs if you specify a value outside this range.
2. X must not be an array element when SRCH is used within an IF/THEN/ELSE statement.
3. If the value of X matches the 256th element, or an element that is a multiple of 256 (512, 768,..), the match is not detected. The value of these elements must be tested separately to detect a match.

SREP\$(A\$,M,B\$,C\$)

SREP\$ returns a string that represents the replacement of B\$ with C\$ in string A\$, starting at position M.

```
10 A$="ABCDEFGHJIJ"  
20 B$="DE"  
30 C$="123"  
40 D$="SREP$(A$,4,B$,C$)"
```

D\$ contains "ABC123FGHIJ."

Note: An error occurs if the string length exceeds 255.

See Programming consideration in "Substring referencing."

Standard format

See "Integer format."

Statement length

See "BASIC statements."

Statement numbers

See "Line numbers."

Statements

See "BASIC statements."

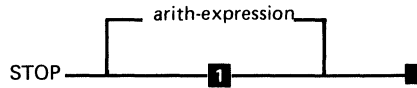
Status line

See "Character set"
"Device sharing"
"DISPLAY"
"Status line" in the *Operator Reference Manual*
System Messages.

STOP

STOP statement

The STOP statement stops the program and closes all files.



1 CODE is 0

The syntax of the STOP statement is as shown above, where:

arith-expression is an expression that is rounded to an integer and is used to set the CODE variable. It is in the range of 0 to 9999. If the parameter is not specified, the default is zero.

Unlike the END statement, the STOP statement can appear anywhere in a BASIC program.

Example

With arith-expression:

```
110 CODEVAL = 139
120 STOP CODEVAL
```

In this example, the STOP statement sets the value returned by CODE to 139 and stops the program.

Without arith-expression:

```
110 STOP
```

In this example, the STOP statement sets the value returned by CODE to 0 (the default value) and stops the program.

Storage use

The following formulas are used to estimate the amount of System/23 internal storage used by unedited programs. The total storage available is indicated by the HELP STATUS command immediately following CLEAR.

For programs, calculate the following items and add 27 bytes for overhead:

Item	Bytes	Notes
Statement	7	
Function reference	2	4
Keyword	1.5	1
Label	1.5	1 4
System function	1.5	1
Variable/array reference	1.5	1 4
Line number reference	4	
Character literal	2	2
Numeric literal	4 or 10	5
Expression	1	3
FORM field specifier	2	
PIC specification	2	2
Subscripts	4 or 6	6
Substring	6	6
Operators	1	7
Punctuation	1	8
Asterisk	2	9
FOR, NEXT	4 or 25	10
CHAIN, USE, LIST	2	11

Storage

Storage use (continued)

- 1** Frequently used functions, variables, and first 63 user defined names are 1 byte. Others are 2 bytes.
- 2** Plus the number of characters.
- 3** See "Arithmetic expression" and "Character expression."
- 4** Name length plus 5 for first-time reference
- 5** Literals used in DIM and FORM use 4 bytes.
- 6** 4 bytes for () format, 6 bytes for (,) format, 6 bytes for (:) format. Byte totals only included punctuation characters and end-of-expression overhead.
- 7** Applies to logical operators, arithmetic operators, concatenation character, and =.
- 8** Punctuation characters are: # : ; () and ,.
- 9** Used in DIM and FORM
- 10** FOR is 4 bytes, NEXT is 25 bytes.
- 11** The variable list and any remark that follows, is treated as a single-character literal string. Add the number of characters.

When the program is run, both objects you define and system objects use storage as follows:

Item	Bytes	Notes
Character variable	7 + current number of characters	1
Character array	13+ (1+DIM length) *number of elements	1
Numeric variable	15	1
Numeric array	13+ (9*number of elements)	1
File controls	132 per file	2
Procedure	650/132 per procedure	3

1 Allocated at first reference, released at CLEAR/RUN

2 Allocated at OPEN, released at CLOSE

3 Allocated at PROC/SUBPROC, released at EOF

As the program is run, temporary results, work areas, and I/O buffers are allocated as needed and automatically released. No single item can exceed 64K bytes. Editing never decreases the size of a program. To recover space when lines are deleted, SAVE SOURCE, LOAD, and SAVE internal again. Storage can be recovered from arrays by redimensioning to one element. See "Redimensioning arrays" The System/23 will reserve a minimum amount of storage for internal use, so your program cannot entirely fill up storage.

Example

```
100 LET WORK5$(1:2)=RUNDAT$(5:6)
```

Item	Bytes	Overhead	Comments
LET	1.5		Keyword
WORK5\$	1.5	11	Variable/array
(:)	6		Substring
1	10		Numeric literal
2	10		Numeric literal
=		1	Operators
RUNDAT\$	1.5	12	Variable/array
(:)	6		Substring
5	10		Numeric literal
6	10		Numeric literal
		7	Statement
		27	Program first line
Totals	57.5	+ 57 =	114.5 bytes

Storage

Storage use (continued)

200 DEF FNA\$(R,K\$)=STR\$(R+5) &K\$

Item	Bytes	Overhead	Comments
DEF	1.5		Keyword
FNA\$	2	9	Function reference
(1		Punctuation
R	1.5	6	Variable/array
,	1		Punctuation
K\$	1.5	7	Variable/array
)	1		Punctuation
=	1		Operators
STR\$	1.5		System function
(1		Punctuation
R	1.5		Variable/array
+	1		Operators
5	10		Numeric literal
	1		Expression
)	1		Punctuation
&	1		Operators
K\$	1.5		Variable/array
	1		Expression
		7	Statement
Totals	31	+ 29 =	60 bytes

STR\$(X)

STR\$ returns the string that is the character representation of the value X. The string has the same appearance as though a PRINT X had been issued. There are no leading or trailing blanks. See "PRINT statement."

X=12


A\$=STR\$(X)

A\$ contains "12."

SUBPROC

SUBPROC command

The SUBPROC command initiates the use of a new procedure file without closing the currently active procedure file. A procedure file is a DISPLAY I/O file that contains BASIC statements, system commands, and input data.

SUBPROC — file-spec — 

The syntax of the SUBPROC command is as shown above, where:

file-spec is the file specification, which consists of a filename followed by an optional VOLID and device address (see "File specification parameter").

The SUBPROC command is identical to the PROC command (see "PROC command"), with the following exceptions:

- The SUBPROC command may be issued from within a procedure file without causing the procedure file to be closed.
- Termination of a procedure invoked by SUBPROC will cause the procedure file input to revert to the invoking procedure.
- The maximum number of procedure files that can be open at one time is five.

INPUT or LINPUT from a procedure will cause an EOF error when the procedure or subprocedure is exhausted.

Subroutines

See "GOSUB statement."
"RETURN statement."

Subscripted Variables

See "Arrays."

Substring referencing

A substring is a part of a string rather than the entire string. Normally, the entire string is referenced. However, sometimes only a part of a string needs to be referenced. The substring reference extracts, replaces, or inserts characters in a character string.

The substring reference denotes the position within a character string by:

character string (arith-expression:arith-expression)

The first arithmetic expression indicates the beginning position, and the second arithmetic expression indicates the ending position of the substring. The arithmetic expressions cannot be negative and are rounded to integers.

The character string can be an array element. For example:

K\$ = ABC\$(5)(A:B)

Substring

Substring referencing (continued)

Substring referencing rules are as follows:

- Rule 1. If the beginning position is less than one, it is considered to be one.
- Rule 2. If the beginning position is greater than the length of the character string, the substring addressed follows the last character of the character string. For example: if A\$ equals "ABCD" the statement A\$(5:7)="123" would result in "ABCD123."
- Rule 3. If the ending position is greater than the length of the character string, it is assumed to be the length of the character string.
- Rule 4. If the beginning position is greater than the ending position, the substring addressed immediately precedes the beginning position character of the character string. The value of the ending position has no significance. For example if A\$ equals "ABCD" then the statement A\$(3:1)="123" would result in "AB123CD."
- Rule 5. In order to assign characters to a string at a specified location, that location must be allocated.

Examples of substring referencing

For the following examples assume that:

A\$="ABCDE"

and

B\$="WXYZ"

Extraction of characters:

<i>Statement</i>	<i>Result</i>
E\$=A\$(2:3)	E\$ equals "BC"
E\$=A\$(4:4)	E\$ equals "D"
E\$=A\$(0:2)	E\$ equals "AB" the 0 is considered to be one (see rule 1)
E\$=A\$(7:8)	E\$ equals "" a null string (see rule 2)
E\$=A\$(4:8)	E\$ equals "DE" the eight is considered to be five (see rule 3)

Substring

Substring referencing (continued)

Replacement of characters:

<i>Statement</i>	<i>Result</i>
A\$(3:4)="12"	A\$ equals "AB12E," "CD" is replaced with "12."
A\$(3:4)=""	A\$ equals "ABE," "CD" is deleted
A\$(3:4)=B\$(1:2)	A\$ equals "ABWXE," "CD" is replaced by "WX"
A\$(3:4)=B\$(1:4)	A\$ equals "ABWXYZE," "CD" is replaced by "WXYZ"

Insertion of characters:

<i>Statement</i>	<i>Result</i>
A\$(1:0)="123"	A\$ equals "123ABCDE," "123" is inserted before the "A" in A\$ (see rule 4)
A\$(1:0)=B\$(3:4)	A\$ equals "YZABCDE," "YZ" is inserted before the "A" in A\$ (see rule 4)
A\$(3:2)=B\$(1:4)	A\$ equals "ABWXYZCDE," "WXYZ" is inserted between the "B" and "C" in A\$ (see rule 4)
A\$(7:8)="123"	A\$ equals "ABCDE123," 123 is inserted after the "E" in A\$ (see rule 2)

Example using subscripts

```
10 ABC$(4) = "ABC"  
20 I = 1  
30 J = 2  
40 K = 4  
50 ABC$(K)(I:J) = "12"
```

Results in ABC\$(4) being "12C"

Example Rule 5

```
10 DIM A$*10  
20 A$(1:4) = "ABCD"  
30 A$(9:10) = "EF"
```

Results in A\$ being "ABCDEF"

Notice that character string "EF" was not assigned to location 9 and 10 of A\$ as specified. To cause "EF" to be assigned to location 9 and 10 of A\$, A\$ must first be allocated 10 characters. To achieve that, insert statement 15 (see rule 5).

```
15 A$ = RPT$(" ",10)
```

which results in "EF" being assigned to location 9 and 10:

```
"ABCD  EF"
```

Substring

Substring referencing (continued)

Programming consideration

(This programming consideration does not apply if the system release code field on your system status line is "1.05.")

On a 64K or larger machine, a string operation which causes the insertion of additional characters, which extends the string length, may cause unpredictable results. These operations are:

A(m:n)=B$$ where $M+LEN(B$)-1>N$

and

$SREP$(A$,M,B$,C$)$ where $LEN(B$)<LEN(C$)$

In these circumstances, the correct results may be obtained by using the following circumvention:

Instead of:

A(M:N)=B$$

use:

A=A$(1:M-1) \&B$ \&A$(N+1:255)$

Instead of:

$SREP$(A$,M,B$,C$)$

Example using subscripts

```
10 ABC$(4) = "ABC"  
20 I = 1  
30 J = 2  
40 K = 4  
50 ABC$(K)(I:J) = "12"
```

Results in ABC\$(4) being "12C"

Example Rule 5

```
10 DIM A$*10  
20 A$(1:4) = "ABCD"  
30 A$(9:10) = "EF"
```

Results in A\$ being "ABCDEF"

Notice that character string "EF" was not assigned to location 9 and 10 of A\$ as specified. To cause "EF" to be assigned to location 9 and 10 of A\$, A\$ must first be allocated 10 characters. To achieve that, insert statement 15 (see rule 5).

```
15 A$ = RPT$(" ",10)
```

which results in "EF" being assigned to location 9 and 10:

"ABCD EF"

Substring

Substring referencing (continued)

Programming consideration

(This programming consideration does not apply if the system release code field on your system status line is "1.05.")

On a 64K or larger machine, a string operation which causes the insertion of additional characters, which extends the string length, may cause unpredictable results. These operations are:

A(m:n)=B$$ where $M+LEN(B$)-1>N$

and

$SREP$(A$,M,B$,C$)$ where $LEN(B$)<LEN(C$)$

In these circumstances, the correct results may be obtained by using the following circumvention:

Instead of:

A(M:N)=B$$

use:

A=A$(1:M-1) &B$ &A$(N+1:255)$

Instead of:

$SREP$(A$,M,B$,C$)$

use:

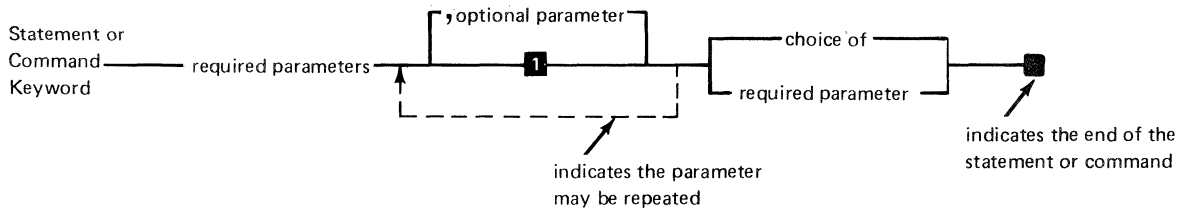
```
10 I=M
20 LOOP: Z=POS(A$,B$,I)
30 IF Z=0 THEN QUIT
40 A$=A$(1:Z-1) &C$ &A$(Z+1:255)
50 I=Z+LEN(C$)
60 GOTO LOOP
70 QUIT:STOP
```


Syntax

Syntax description

When syntax formats are described in this manual, capitalized expressions, lowercase expressions, and special characters (such as a comma, colon, exclamation point, or an asterisk) have special meaning.

Syntax of the BASIC commands and statements is presented in the following format:



Where:

Statement or Command keyword is a BASIC statement such as LET or a command such as RUN.

required parameter is an item that must be included such as the line reference in GOTO 100.

optional parameter is an item that may be included if desired such as ELSE in an IF, THEN, ELSE statement.

indicates that the parameter may be repeated means that more than one parameter can be included such as the variables in INPUT A, B, C ...

choice of required parameters means that one of the parameters must be included such as the choice between numeric or character constants in a DATA statement.

indicates the end of the statement or command refers to the block that indicates the end of the syntax.

To read the syntax of a command or statement, read from left to right along the main line. When you reach an optional parameter, you can either include that parameter or continue along the main line. When you reach a choice of required parameters, you must include one of the parameters with your command or statement.

Syntax

Syntax description (continued)

If a parameter is shown in uppercase letters, you must enter it exactly as it appears. You must also enter any special character (such as a comma or colon) that appears in the diagram.

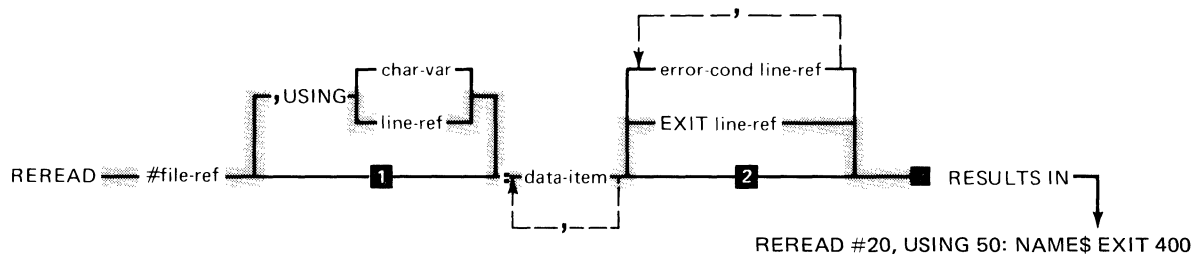
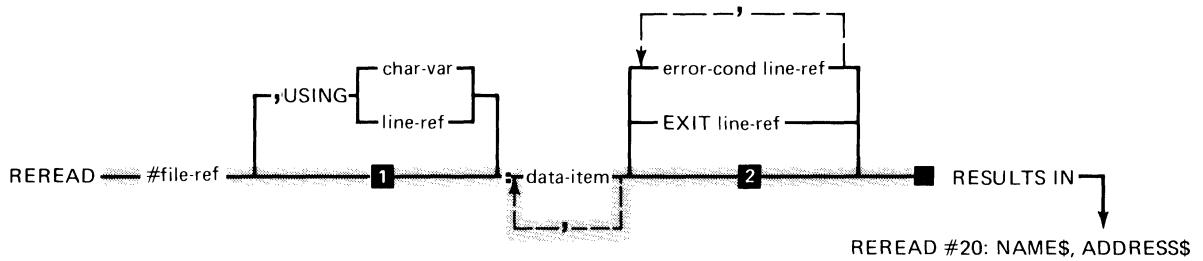
All lines entered in BASIC program entry mode are converted to English uppercase prior to syntax checking.

To prevent remarks or character data on DATA statements from being converted to English uppercase, they must be enclosed in quotation marks.

If you do not include an optional parameter, the System/23 provides a default value or action. The defaults are listed in the description of the statement or command. The syntax diagrams include a number (such as **1**) that corresponds to the defaults listed.

In the case of the MERGE, REPLACE, and VOLID commands *only*, you must include a comma to indicate that you have omitted an optional parameter.

Here are two examples using the REREAD statement:



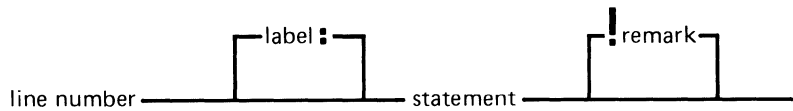
In these examples, you must include the *file-ref* parameter following the keyword REREAD. You may choose to include the USING parameter in which case you must also include either the *char-var* or *line-ref* parameter. You must include the colon, followed by at least one *data-item*. Note that you may list more than one *data-item*. You may choose to include either *EXIT line-ref* or *error-cond line-ref*.

In the first example, the optional parameters are omitted. Therefore the default actions are taken.

Syntax

Syntax description (continued)

The syntax for a BASIC statement is as shown:



A keyword in a BASIC statement or system command must be followed by a blank except where a comma, parenthesis, or other appropriate delimiter is defined. Also a blank must follow the leading line number in a BASIC statement.

A label can be added to any BASIC statement except a DEF statement (see "Labels").

A remark can be added at the end of any system command or BASIC statement except a DATA statement (see "Remarks").

SYSTEM command

The SYSTEM command lets you specify special functions for the communications feature.

SYSTEM ————— ■

System commands

The system commands are used for program management, execution, operations, and to control diskette, disk, and printer operations. System commands are instructions that the computer executes immediately. Commands are not part of a BASIC program and do not have line numbers. Commands may be entered either character-by-character from the keyboard and then executed by pressing the Enter key or, by holding down the Cmd key and pressing the appropriate key for the keyword you want. Using the Cmd key inserts the keyword by pressing a single key, providing faster operation. It also prevents typing errors. It should be noted that commands can be executed from the keyboard or from a PROC, but not from a program. The commands direct the system to perform the following operations.

- Program execution—Start or resume execution of a BASIC program, procedure, or Customer Support Function.
- Program management—Load or save programs or data on a diskette or on the disk. Display program status (name and storage).
- Program operation—List, edit, and renumber program statements or merge several programs into one.
- File management—Lists, renames, protects, drops, or frees files on a diskette or on the disk.
- Set date and time.
- PRINT variables or expression results.
- Assign values to variables.

System commands

System commands (continued)

Parameters required for a command can be entered on the line after the command keyword. The command operation starts after the Enter key is pressed. The command keywords and their functions are:

ALERT	Alerts the operator from a procedure file
AUTO	Assigns line numbers
CLEAR	Deletes data or program from work area
DATE	Sets the DATE\$ variable
DEL	Delete lines of a BASIC program or data work area
DIR	Lists a file directory
DROP	Removes file data
FREE	Eliminates a file
GO	Resumes interrupted processing
HELP STATUS	Displays the name of the work area
LET	Assigns a value to a variable
LINK	Loads and executes Customer Support Functions
LIST	Displays a BASIC program or data file work area
LISTP	Prints a list of lines in the work area
LOAD	Loads a BASIC program or data file
MERGE	Merges a BASIC program and source file
PRINT	Displays data on screen or printer
PROC	Initiates command input from a procedure file
PROCERR	Directs system error handling
PROTECT	Write-protects a file, removes share restrictions, or closes a file

RENAME	Renames a file
RENUM	Rennumbers lines
REPLACE	Saves a program to an existing file
RUN	Runs a BASIC program
SAVE	Saves a BASIC program or data file
SKIP	Skips records within a procedure file
SORT	Executes SORT file
SUBPROC	Initiates input from a sub-procedure
SYSTEM	Specifies special communications feature functions
TIME	Sets time of day
VOLID	Lists or changes a volume-identification or access status

System functions

System functions

The System/23 BASIC language includes system functions (intrinsic functions) that perform a number of commonly used operations. In addition, the function can be defined and named by using the DEF statement (see “DEF statement”).

The system functions can be used anywhere in a BASIC expression in which constants, variables, or array element references can be used. See “Arithmetic expressions”

Some of the functions have one or more arguments that produce a single result. An invalid argument will cause an error.

System function rules are as follows:

- Arithmetic expressions are indicated by X or M.
- Character scalar arguments are indicated by A\$, B\$, or C\$.
- File reference numbers are indicated by N and can be arithmetic expressions. Noninteger values are rounded.

Note: The arguments to system functions can be expressions that include function references.

See “Program 4—Sample” in Appendix A.

ABS(X)	Absolute value of X
AIDX(array name)	Ascending index of the source array
ATN(X)	ARC tangent of X
CEIL(X)	Next larger number
CHR\$(X)	Position in native collating sequence
CON	Sets array to a constant and redimensions
COS(X)	Cosine of X
DIDX(array name)	Descending index of the source array
DISPLY(X)	Current screen display page
EXP(X)	E raised to X power
FILE(N)	Status of the file
FILE\$(N)	File specification
FREESP(N)	Space available
INT(X)	X or next smaller number
HEX\$(A\$)	Character equivalent of hexadecimal value
KLN(N)	Key length of file N
KPS(N)	Key position of file N
LEN(A\$)	Length of A\$
LOG(X)	Natural log of X
LPAD\$(C\$,X)	Pad with blanks on left
LTRM\$(C\$)	Trim blanks from left
MAX(X1,X2,...)	Maximum value of list
MIN(X1,X2,...)	Minimum value of list
ORD(A\$)	Collating location of A\$
PIC\$(C\$)	Return or set PIC currency symbol

System functions

System functions (continued)

POS(A\$,B\$,X)	Substring location
REC(N)	Last record number used in file N
RLN(N)	Record length
RND(X)	Random number
ROUND(X,M)	Rounded value of X
RPAD\$(A\$,X)	Pad with blanks on right
RPT\$(A\$,M)	Repeat string A\$
RTRM\$(A\$)	Trim blanks from right
SGN(X)	Sign of X
SHIFT(X)	Returns machine type or sets shift mode
SIN(X)	Sine of X
SQR(X)	Square root of X
SRCH	Search table
SREP\$	Replaces substring
STR\$(X)	Character string representation of X
TAN(X)	Tangent of X
UDIM(array,X)	Highest subscript of array
VAL(A\$)	Numeric equivalent of numeric representation
WSID\$	Shared 5246 and 5247 connector number
ZER	Zero array and redimension

System keywords

See “System commands” and “Reserved words.”

System variables

The system variables are used by the system to aid in time stamping, program control, and error recovery. System variables are maintained by the system. They cannot be the target of an assignment, nor can they be referenced in a substring operation. Otherwise, they may be used in any context where a user variable is allowed.

The following are system variables that are set to arithmetic values:

- **CMDKEY**—INPUT/LINPUT termination code
- **CNT**—I/O variable count
- **CODE**—Program termination code
- **ERR**—Error code
- **FILENUM**—Last file causing an error
- **LINE**—Last program line number causing an error
- **PROCIN**—Is RUN PROC active?

The following are system variables that are set to character values:

- **DATE\$**—Value set by DATE command
- **KSTAT\$**—Last key stroke
- **TIME\$**—Current time of day
- **WSID\$**—5246 and 5247 attachment code

TAB

TAB function

See "PRINT statement."

Tables

See "Arrays."

TAN(X)


TAN returns the tangent of X, where X is in radians and is less than 1E10. Specify a value for X greater than $-(\pi/2)$ or less than $\pi/2$ for best accuracy.

THEN

See "IF, THEN, ELSE statement."

TIME command

The TIME command is used to enter the time of day into the system. The system variable TIME\$ is set to the value specified.

TIME — hh:mm:ss 

The syntax of the TIME command is as shown above, where:

hh:mm specifies the time in hours, minutes, and seconds.

The time is set to 00:00:00 when you switch on power.

The value of TIME\$ wraps on 23:59:59 to 00:00:00. The DATE\$ variable is not incremented.

TIME\$

TIME\$ returns an eight-character string that is initialized by the TIME command and maintained by the system. When you switch power on, the value is set to 00:00:00.

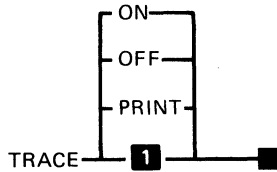
Tips and techniques

See Appendix C, "Performance tips and techniques."

TRACE

TRACE statement

The trace statement traces all or part of a program's execution.



1 ON to display

The syntax of the TRACE statement is as shown above, where:

ON specifies that tracing is to be displayed on the screen. This is the default.

OFF specifies that tracing is to be stopped.

PRINT specifies that tracing is to be printed on the system printer. PRINT cannot be used if device address 10 is OPEN to a file reference number other than #255. If TRACE PRINT is active, device 10 cannot be OPEN to any file reference number except #255 (see "Printer assignment").

To TRACE an entire program without modifying it, see "RUN command" (TRACE option).

Trim

See "LTRM\$"
"RTRM\$"

UDIM (array,X)

UDIM returns the value of the upper dimension X of the array, where X is either an integer 1 or an integer 2.

X=1 returns the row dimension
X=2 returns the column dimension

An error occurs if X is neither 1 nor 2, or if 2 is specified and the array is one-dimensional.

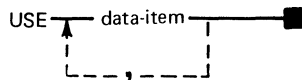
```
10 DIM A(5,3)
20 B = UDIM(A,1)
30 C = UDIM(A,2)
```

B contains 5 (row dimensions).
C contains 3 (column dimensions).

USE

USE statement

The USE statement lists the variables passed from one program to another during chaining.



The syntax of the USE statement is as shown above, where:

data-item is the variable or array (without the keyword MAT) passed to the chained-to program. The list of data items is not syntax checked until the USE statement is executed.

The list of data items specified on the USE statement must exactly match those listed on the corresponding CHAIN statement. If an array or a character variable is passed during chaining, the array or the character variable must be dimensioned in the chained-to program to the same number of elements and string size to which it was dimensioned in the chained-from program. The options of the chained-to program must be identical to the options of the chained-from program.

Only one USE statement is permitted in a program.

VAL(A\$)

The numeric value associated with the string A\$ is returned. If the conversion of the numeric representation results in a value that causes an underflow, then the value returned is zero. If the conversion of the numeric representation results in a value that causes an overflow, then the value returned is the largest number. A conversion error occurs if: A\$ is not a valid numeric representation or if the variable in parentheses is a scalar that has not been set.

Variable names

All variable names must be unique. You cannot use the same name to designate an array, a variable, a function, or a label.

There are three types of variable names. They are:

- Numeric variable or numeric array name
- Character variable or character array name
- Function names

Numeric variable or numeric array name

A numeric variable name or a numeric array name must start with an alphabetic character, followed by up to seven alphabetic or numeric characters. The name must be surrounded by blanks, commas, parentheses, or other delimiters as shown in the syntax. For example:

```
10 SALARY = 850  
20 TAXRATE(3) = .22
```

Variable names

Variable names (continued)

Character variable or character array name

A character variable name or character array name must start with an alphabetic character, followed by up to seven alphabetic or numeric characters and ending in a \$. The name must be surrounded by blanks, commas, parentheses, or other delimiters as shown in the syntax. For example:

```
20 DIM ADDRESS$*20
30 ADDRESS$ = "22136 LARKSPUR TRAIL"
40 NAME$(3) = "SCOTT"
```

Function names

Function names must start with FN, followed by an alphabetic character, followed by up to seven alphabetic or numeric characters. The function name must end in a \$ only if a character result is returned (see "DEF, FNEND statement"). For example:

```
50 DEF FNFICA(X) = .0613*X
60 DEDUCT = FNFICA(SALARY)
70 DEF FNCONNECT$(X$, Y$) = X$ & Y$
80 PRINT FNCONNECT$(NAME$(3), ADDRESS$)
```

Note: Names which are already used in the BASIC language may not be used as variable names. All words beginning with FN are reserved words (see "Reserved words").

Variables, arithmetic

See "Arithmetic variables."

Variables, character

See "Character variables."

Variables, internal

See "System variables."

six alphabetic or numeric characters. *New-volume id* is only necessary if the volume identification is being changed.

owner-id specifies the owner identification for the disk volume or diskette. This entry can be up to 14 alphabetic or numeric characters. *Owner-id* is only necessary when the owner identification is being changed or when the OFF parameter is used.

ON specifies that the protection indicator for the volume is to be turned on, making the volume inaccessible.

OFF specifies that the protection indicator for the volume is to be turned off, making the volume accessible.

Do not change the VOLID of a disk volume or diskette with files that are open and in use. An error message results if you do so when using the 5247 Disk. No error message results for diskette files.

VOLID

VOLID command (continued)

Example

```
VOLID DEBITS,DEBTS,ZEPOL,ON
```

In this example, the volume identification of the volume named DEBITS is changed to DEBTS. The owner identification is changed to ZEPOL. The volume protection indicator is turned on.

```
VOLID DEBITS,,NEW
```

In this example, the owner identification of the volume DEBITS is changed to NEW. Notice that no *new-volume-id* is entered because the volume identification is not changed. However, a comma must still be entered as shown in the example. The extra comma indicates that a parameter is being skipped.

```
VOLID  
1 MASTER PAYROLL
```

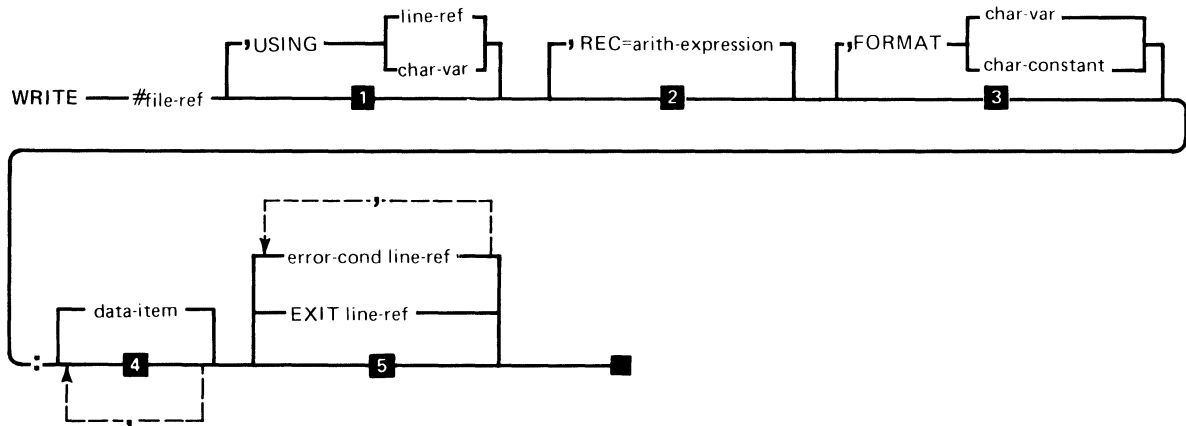
In this example, the VOLID command is entered alone and the volume ID and owner ID are displayed for the diskette in drive 1.

Work area

See "Storage use."

WRITE statement

The WRITE statement replaces a deleted record or adds a record to an internal file. The file may be opened for sequential, relative, or keyed access.



- 1 Unformatted write
- 2 Sequential or keyed file only. Add to the end of the file
- 3 Communications special control functions not in use
- 4 WRITE record with no data
- 5 Interrupt on error unless ON is active

The syntax for the WRITE statement is as shown, where:

file-ref is a numeric expression. See "File reference parameter."

USING specifies a line reference of a FORM statement or a character variable containing a FORM statement. *line-ref* can be a line number or label. The FORM statement is used to indicate the representation and location of the variables in the output.

WRITE

WRITE statement (continued)

REC= specifies the record having a record number equal to an arithmetic expression. This parameter must be used when *RELATIVE* is specified in the *OPEN* statement. When replacing a record, the record number refers to the deleted record. When adding a record, the record number is $n + 1$ (n is the total number of records in the file).

FORMAT is a communications feature clause. It specifies that special control functions are requested. The control functions can be specified as a *char-constant* (character constant) or in a *char-var* (character variable).

data-item specifies the names of variables or expressions to be written to the file. The data-item can include variables, array elements, entire arrays (preceded by *MAT*), or numeric or character expressions. Data-item must be separated by a comma and must be preceded by a colon.

error-cond line-ref specifies the line number or label that the program should transfer to if one of the error conditions occurs. The following error conditions may be included in any order:

- *CONV*—Conversion error
- *DUPREC*—Record already exists
- *EOF*—End of volume
- *IOERR*—Input/output error
- *NOREC*—Invalid record reference
- *SOFLOW*—String overflow

EXIT specifies the line number or label of an *EXIT* statement that the system references if an error occurs.

For related information, see "Internal I/O file formatting" "READ statement" "OPEN statement" "FORM statement" and I/O tables in Appendix B.

Programming considerations

- Added keyed records
 - The record is added to the master file and a pointer to it is added to the index file. Running the Index Customer Support Function is recommended to speed up subsequent file access. If the new index file record was created and the master file write is unsuccessful, the index file will contain an invalid entry. The condition will cause a NOREC error (no record found) on a subsequent READ KEY= for the record. To prevent the addition of a duplicate key, precede the WRITE with a READ KEY= and check for a NOKEY condition. The specification of no duplicate key in the Index Customer Support Function does not prevent the addition of duplicate keys.
- CLOSE statement
 - Execution of the WRITE statement does not always result in an immediate physical write to the diskette or disk. To ensure that it will, a CLOSE can be issued. When a file is opened with SHR specified, only one program can write while the other program reads. When a disk file is opened with SHRU specified, the physical write is done immediately.
- Unspecified record locations
 - Unspecified record locations are written as blanks if a USING clause was present.
- KEY/SEARCH
 - The KEY/SEARCH position specification cannot be used.

WRITE

WRITE statement (continued)

- OPEN statement
 - OUTPUT or OUTIN must be specified on the OPEN INTERNAL statement.
- No data
 - If the I/O list is omitted, there will be no transfer of data from variables. A record will be written.

See "Program 5—Sample" in Appendix A.

WSID\$

WSID\$ is A BASIC variable that allows programs to determine to which I/O connector of the 5246 Diskette Unit and the 5247 Disk Unit the computer is attached.

WSID\$ is two characters long. The first character gives information about the 5247 Disk Unit and the second character gives information about the 5246 Diskette Unit.

First character

- 0 disk not attached to the computer or not operational.
- 1 computer attached to disk connector 1.

Second character

- 1 diskette unit not attached to the computer, not operational, or the computer is attached to diskette unit connector 1.

- 2 computer attached to disk connector 2.
- 2 computer attached to diskette unit connector 2.
- 3 computer attached to disk connector 3.
- 4 computer attached to disk connector 4.

If you do not have a 5247 Disk Unit, the only possible values for WSID\$ are: 01 and 02.

WSID\$ is set when the computer is turned on and does not change until the computer is turned off. The next time you turn the computer on, the WSID\$ will reflect the status of the disk and diskette unit at that time. For example:

If the computer is attached to disk connector 3 and diskette unit connector 2, the WSID\$ is 32.

If the diskette unit is switched off when the computer is switched on, the WSID\$ is 31.

You can create a workspace local to a particular computer by concatenating a VOLID with the WDIS\$. For example:

```
Let A$="Name=FILEA/VOL" &WSID$
```

WSID\$

WSID\$ (continued)

If you open a file with the WSID\$ added to the filename, and the WSID\$ is 32, the result is "FILE.NAME32.." If the WSID\$ changes (as in the example) to 31, the same program would open the file as "FILE.NAME31" and a "file not found" condition results. This happens because the file was originally opened as "FILE.NAME32.."

Programming considerations

- Use WSID\$ on temporary files that you create at open and delete at close.
- For permanent files, use only the portion of the WSID\$ that refers to the device the file is stored on.
- Do not include WSID\$ in the filename when you create a file that you migrate between devices.

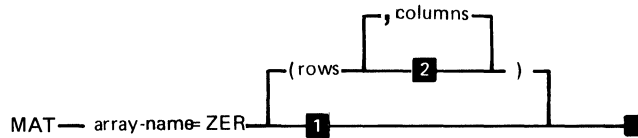
XREF

See "LIST label" under "LIST,LISTP command."

ZER and CON

ZER and CON are functions provided for matrixes.

ZER syntax



- 1** No redimensioning
- 2** Redimension to a one-dimensional array (vector)

The syntax for ZER is as shown, where:

array-name is the name of the array to be set to zero.

ZER sets all the elements of the array to zero.

row, *columns* are the redimensioning specifications for the array. Results are unpredictable if subscripted values are specified for rows and/or columns.

ZER and CON

ZER and CON (continued)

Example

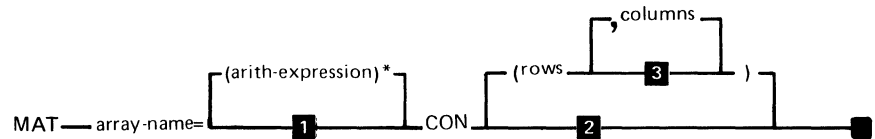
Assume array A is an array with four rows and four columns.

```
10 OPTION BASE 1
20 DIM A(4,4)
  .
  .
  .
250 MAT A=ZER(3,3)
```

Array A before Array A after

1 2 3 4	0 0 0
2 3 4 5	0 0 0
3 4 5 6	0 0 0
4 5 6 7	

CON syntax



- 1** 1
- 2** No redimensioning
- 3** Redimension to one-dimensional array (vector)

The syntax for CON is as shown, where:

array-name is the name of the array to receive the constants.

arith-expression is a scalar arithmetic expression, which must be enclosed in parentheses.

CON sets all the values of *array-name* to the value of the *arith-expression*. If no *arith-expression* was specified, the default is 1.

rows, *columns* are the redimensioning specifications for the array. Results are unpredictable if subscripted values are specified for rows and/or columns.

ZER and CON

ZER and CON (continued)

Example

Assume that array B is an array with five rows and five columns.

```
10 OPTION BASE 1
20 DIM B(5,5)
  .
  .
  .
250 MAT B=(3*2)*CON(3,4)
```

Array B before	Array B after
1 2 3 4 5	6 6 6 6
2 3 4 5 6	6 6 6 6
3 4 5 6 7	6 6 6 6
4 5 6 7 8	
5 6 7 8 9	

Appendix A. Sample programs

Program 1 - Sample

```
00010 PRINT #255:"SAMPLE 1 PROGRAM";TAB(1)
00020 TRACE PRINT ! .          TRACE, DISPLAY LINE NUMBERS WITH RESULTS
00030 PRINT #255:ABS(5),ABS(-5) ! ABS, ABSOLUTE VALUE SYSTEM FUNCTION
00040 PRINT #255:ATN(1) ! .     ARC TANGENT SYSTEM FUNCTION
00050 ! 4 ROW, 4 COLUMN ARITHMETIC ARRAY A, CHARACTER ARRAY C#
00060 ! (MAXIMUM STRING LENGTH 10)
00070 DIM A(3,3),C$(3,3)*10
00080 LET A(2,2)=5 ! .         ASSIGN NUMERIC ARRAY ELEMENT
00090 PRINT #255:A(2,2) ! .     PRINT NUMERIC ARRAY ELEMENT
00100 LET C$(2,2)="XYZ" ! .     ASSIGN NUMERIC ARRAY ELEMENT
00110 PRINT #255:C$(2,2) ! .     PRINT CHARACTER ARRAY ELEMENT
00120 PRINT #255:MAT A ! .     PRINT ENTIRE ARRAYS A AND C#
00130 PRINT #255:MAT C# ! .     PRINT ENTIRE ARRAYS A AND C#
00140 PRINT #255:9.E+126,1.E-126 ! PRINT MAXIMUM AND MINIMUM NUMBERS
00150 PRINT #255:1.123456789012345 ! PRINT FULL SIGNIFICANCE
00160 DATA 266,266.,266.00,.266E3,.266E+3,+.266E3,+.266E+3
00170 DATA +0.266E+003,+2.66E2,26.6E1,2660E-1,+26600E-2,.00266E5
00180 DIM N(2,3) ! .          DIM ARRAY FOR REPRESENTATION EXAMPLE
00190 READ MAT N ! .          READ DIFFERENT VERSIONS OF 266 INTO ARRAY N
00200 PRINT #255:MAT N ! .     PRINT RESULTS
00210 PRINT #255:1+2,1*2*3,(1+2)*3,(1+2)*(3+4)
00220 PRINT #255:1+2*3+4,1+2*3/4,(1+2)*(3/4),TAB(0),1+2*3/4,2**3
00230 PRINT #255:50+10**2/(2*(13+12))-2
00240 LET PAYMENT=122.3 ! .     ASSIGN NUMERIC VARIABLE
00250 LET TODAY=30
00260 LET VALUE=20
00270 PRINT #255:PAYMENT,VALUE,TODAY
00280 PRINT "PRESS INQ TO CONTINUE"
00290 ON ATTN GOTO ENDING ! .   INQ KEY CAUSES TRANSFER TO ENDING
00300 TRACE OFF ! .           STOP TRACE WHILE WAITING FOR INQ
00310 LET Z=0 ! CANNOT DETECT INQ ON A GOTO; MUST HAVE SOMETHING ELSE IN LOOP
00320 GOTO 310
00330 ENDING: ON ATTN IGNORE ! . INQ KEY IGNORED
00340 TRACE PRINT ! .         RESTART TRACE
00350 PRINT #255:CEIL(1.2);CEIL(-1.2),CEIL(5);CEIL(-5)
00360 LET D$="ABCD" ! .       INITIALIZE CHARACTER VARIABLE D#
00370 PRINT #255:D$,D$&"EF",D$(2:3) ! PRINT CHARACTER EXPRESSIONS
00380 LET D$(2:3)="XY" ! .     REPLACE "BC" WITH "XY"
00390 PRINT #255:D$
00400 LET D$(2:3)="12345" ! .   REPLACE "XY" WITH "12345"
00410 PRINT #255:D$
00420 LET D$(2:6)="" ! .       REPLACE 5 CHARACTERS WITH NULL
00430 PRINT #255:D$
```

Sample programs

Program 1 - Printed output

SAMPLE 1 PROGRAM

```
00030 5 5
00040 .785398163397448
00080 00090 5
00100 00110 XYZ
00120
0 0 0 0
0 0 0 0
0 0 5 0
0 0 0 0
00130

XYZ

00140 9.E+126 1.E-126
00150 .123456789012345
00190 00200
266 266 266 266
266 266 266 266
266 266 266 266
00210 3 6 9 21
00220 11 2.5 2.25
2.5 8

00230 50
00240 00250 00260 00270 122.3 20 30
00280 00290 00300 00350 2 -1 5 -5
00360 00370 ABCD ABCDEF BC
00380 00390 AXYD
00400 00410 A12345D
00420 00430 AD
```

Program 2 - Sample

```
00010 PRINT #255:"SAMPLE 2 PROGRAM";TAB(1)
00020 TRACE PRINT
00030 PRINT HEX$("04");"HIGHLIGHT";HEX$("07");"BLINK";HEX$("08");"REVERSE";;
00040 PRINT HEX$("0A");"HIGHLIGHT,BLINK";HEX$("0B");"REVERSE,BLINK";HEX$("14")
00050 DIM E$*255
00060 TRACE OFF
00070 FOR I=64 TO 255 ! USE CHR$ TO GENERATE ALL LETTER GRAPHICS
00080 LET E$=E$&CHR$(I) ! CATENATE EACH CHARACTER TO ACCUMULATED E$
00090 NEXT I
00100 PRINT NEWPAGE ! CLEAR SCREEN
00110 PRINT FIELDS "3,2,C 78":"DISPLAY FOR EACH SETTING OF THE DISPLY FUNCTION"
00120 PRINT FIELDS "5,2,C 64":E$(1:64) ! PRINT CONTENTS OF E$ (ALL CHARACTERS)
00130 PRINT FIELDS "7,2,C 64":E$(65:128)
00140 PRINT FIELDS "9,2,C 64":E$(129:182)
00150 FOR I=1 TO 5 ! SET DISPLY AND PRINT DISPLY SETTING; WAIT FOR ENTER
00160 PRINT FIELDS "15,2,C 78":"DISPLY FUNCTION SETTING IS:"&STR$(DISPLY(I))
00170 PRINT FIELDS "17,2,C 78":"PRESS ENTER TO CONTINUE"
00180 INPUT FIELDS "18,2,C 1":Z$
00190 NEXT I
00200 TRACE PRINT
00210 PRINT #255:"COSINE OF 1 RADIAN IS ";COS(1)
00220 DIM F(4),AINDEX(4),DINDEX(4)
00230 FOR I=0 TO 4 ! FILL F WITH RANDOM NUMBERS
00240 LET F(I)=RND
00250 PRINT #255;
00260 NEXT I
00270 MAT AINDEX=AIDX(F) ! ASCENDING INDEX OF F
00280 MAT DINDEX=DIDX(F) ! DESENDING INDEX OF F
00290 FOR I=0 TO 4
00300 PRINT #255;
00310 PRINT #255,USING 330:F(I),AINDEX(I),DINDEX(I),F(AINDEX(I)),F(DINDEX(I))
00320 NEXT I
00330 FORM N 17.15,X 2,N 1,X 2,N 1,X 2,N 17.15,N 17.15,SKIP 2I
00340 PRINT #255:"TAN(1)=";TAN(1),"TAN(PI/4)=";TAN(PI/4)
00350 PRINT #255:"CHARACTER AND NUMERIC DATA ";STR$(123.45)
00360 END
```

Sample programs

Program 2 - Printed output

SAMPLE 2 PROGRAM

```
00030 00040 00060 00210 COSINE OF 1 RADIAN IS .540302305868036
00230 00240 00250
00260 00240 00250
00260 00240 00250
00260 00240 00250
00260 00240 00250
00260 00270 00280 00290 00300
00310 .131537788143166 0 1 .131537788143166 .755605322195030

00320 00300
00310 .755605322195030 4 3 .218959186328090 .532767237412170

00320 00300
00310 .458650131923449 2 2 .458650131923449 .458650131923449

00320 00300
00310 .532767237412170 3 4 .532767237412170 .218959186328090

00320 00300
00310 .218959186328090 1 0 .755605322195030 .131537788143166

00320 00340 TAN(1)= 1.5574077246552          TAN(PI/4)= .99999999999067
00350 CHARACTER AND NUMERIC DATA 123.45
00360
```


Sample programs

Program 3 - Sample

```
00010 OPTION BASE 1 ! SAMPLE3, DISKETTE FILE SIZE
00020 DIM FILETYPE$(6)*2
00030 DATA BX,HX,04,05,07,08
00040 READ MAT FILETYPE$
00050 PRINT NEWPAGE
00060 PRINT FIELDS "2,2,C 36":"ENTER FILE TYPE (BX,HX,04,05,07,08)"
00070 INPUT FIELDS "2,39,C 2":FTYPE$
00080 ON 1+SRCH(FILETYPE$,FTYPE$,1) GOTO 70,TBX,THX,T04,T05,T07,T08 NONE 70
00090 TBX: GOSUB INREC
00100 LET BYTES=128*RECORDS
00110 GOTO REPORT
00120 THX: GOSUB INREC
00130 LET BYTES=256*RECORDS
00140 GOTO REPORT
00150 T04: GOSUB INREC
00160 PRINT FIELDS "6,2,C 12":"RECORD SIZE"
00170 INPUT FIELDS "6,15,N 4":RSIZE
00180 IF RSIZE>4095 THEN GOTO 170
00190 LET BYTES=(1+RSIZE)*RECORDS
00200 GOTO REPORT
00210 T05: PRINT FIELDS "4,2,C 15":"NUMBER OF LINES"
00220 INPUT FIELDS "4,18,N 5":NLINES
00230 PRINT FIELDS "6,2,C 30":"AVERAGE LINE LENGTH (1 TO 255)"
00240 INPUT FIELDS "6,34,N 3":LINELGTH
00250 LET BYTES=LINELGTH*(1+NLINES)
00260 GOTO REPORT
00270 T08: !
00280 T07: GOSUB INREC
00290 PRINT FIELDS "6,2,C 20":"KEY LENGTH (1 TO 28)"
00300 INPUT FIELDS "6,24,N 2":KEYLGTH
00310 LET BYTES=512*CEIL(RECORDS/INT(512/(KEYLGTH+4)))-1)
00320 IF FTYPE$="07" THEN GOTO REPORT
00330 PRINT FIELDS "8,2,C 21":"NUMBER OF NEW RECORDS"
00340 INPUT FIELDS "8,24,N 8":NEWRECS
00350 LET BYTES=BYTES+512*CEIL(NEWRECS/INT(510/(KEYLGTH+10)))
00360 GOTO REPORT
00370 !
00380 REPORT: PRINT FIELDS "10,2,C 30":"FILE SIZE = "&STR$(BYTES)
00390 PRINT FIELDS "22,2,C 16":"AGAIN ? (YES/NO)"
00400 INPUT FIELDS "22,20,C 3":ANSWER$
00410 IF ANSWER$="YES" THEN GOTO 50
00420 PRINT NEWPAGE
00430 STOP
00440 INREC: PRINT FIELDS "4,2,C 18":"NUMBER OF RECORDS"
00450 INPUT FIELDS "4,21,N 8":RECORDS
00460 RETURN
00470 END
```

Program 3 - Display output A

ENTER FILE TYPE (BX,HX,04,05,07,08) 08

NUMBER OF RECORDS 800

KEY LENGTH (1 TO 28) 14

NUMBER OF NEW RECORDS 200

FILE SIZE = 19456

AGAIN ? (YES/NO)

RUN

FIELDS

1.01

1 2

Sample programs

Program 3 - Display output B

ENTER FILE TYPE (BX,HX,04,05,07,08) HX

NUMBER OF RECORDS 200

FILE SIZE = 51200

AGAIN ? (YES/NO)

RUN

FIELDS

1.01

1 2

Program 4 - Sample

```
00010 PRINT #255:"SAMPLE 4 PROGRAM";TAB(1)
00020 TRACE PRINT
00030 DEF FNTIMESTMP#*30
00040 LET FNTIMESTMP#="DATE: "&DATE#&"," TIME: "&TIME#
00050 FNEED
00060 PRINT #255:FNTIMESTMP#
00070 DEF FNTEST1(X)
00080 LET FNTEST1=X+FNTEST2(X)
00090 FNEED
00100 DEF FNTEST2(Y)
00110 LET FNTEST2=Y*Y
00120 FNEED
00130 PRINT #255:FNTEST1(3) ! RESULT IS 12
00140 DEF FNTEST3(Z)=Z+2
00150 PRINT #255:FNTEST3(6) ! RESULT IS 8
00160 PRINT #255:EXP(1);EXP(2);EXP(.5)
00170 PRINT #255:INT(1.5);INT(-1.5);INT(10)
00180 PRINT #255:LOG(1);LOG(EXP(1));LOG(10)
00190 LET A#=" ABC"
00200 PRINT #255:A#;A#;LEN(A#)
00210 PRINT #255:LTRM$(A#);LTRM$(A#),LEN(LTRM$(A#))
00220 LET B#="XYZ"
00230 PRINT #255:B#;B#;LEN(B#)
00240 PRINT #255:RTRM$(B#);RTRM$(B#),LEN(RTRM$(B#))
00250 LET C#="DEF"
00260 PRINT #255:C#;C#;LEN(C#)
00270 PRINT #255:LPAD$(C#,10);LPAD$(C#,10),LEN(LPAD$(C#,10))
00280 PRINT #255:RPAD$(C#,10);RPAD$(C#,10),LEN(RPAD$(C#,10))
00290 PRINT #255:RPT$("GHI",3)
00300 PRINT #255:MIN(1,3,-5),MIN(-5,1,3)
00310 PRINT #255:POS ("ABCDCDE","DE",1)
00320 PRINT #255:POS ("ABCDCDE","CD",4)
00330 LET A=123.456789
00340 PRINT #255:A;ROUND(A,3);ROUND(A,0);ROUND(A,-2)
00350 PRINT #255:SGN(-3),SGN(0),SGN(23.2)
00360 PRINT #255:SIN(PI);SIN(2*PI);SIN(PI/2)
00370 PRINT #255:SQR(4);SQR(9);SQR(2)
00380 PRINT #255:ORD("A"),ORD("1")
00390 PRINT #255:SREP$("ABCDEFG",3,"AB","XY")
00400 PRINT #255:10+VAL("12")
00410 END
```

Sample programs

Program 4 - Printed output

```
SAMPLE 4 PROGRAM  
00030 00060 00040 00050 DATE: / / , TIME: 00:02:25  
00070 00100 00130 00080 00110 00120 00090 12  
00140 00150 8  
00160 2.718281828459 7.389056098931 1.6487212707  
00170 1 -2 10  
00180 0 .9999999999999983 2.30258509299405  
00190 00200 ABC ABC 6  
00210 ABCABC 3  
00220 00230 XYZ XYZ 6  
00240 XYZXYZ 3  
00250 00260 DEFDEF 3  
00270 DEF DEF 10  
00280 DEF DEF 10  
00290 GHIGHIGHI  
00300 -5 -5  
00310 6  
00320 5  
00330 00340 123.456789 123.457 123 100  
00350 -1 0 1  
00360 0 -1.E-14 1  
00370 2 3 1.414213562373  
00380 193 241  
00390 ABXYCDEFG  
00400 22  
00410
```

Program 5 - Sample

```
00010 PRINT #255:"SAMPLE 5 - OPEN, CLOSE(FREE), INPUT, LINPUT"
00020 PRINT #255:"          READ, REREAD, WRITE, REWRITE";TAB(1)
00030 TRACE PRINT
00040 OPEN #1:"NAME=TEST.DISP/SAMPLE,SIZE=512",DISPLAY,OUTPUT
00050 PRINT #1:"LINE,1"
00060 PRINT #1:"LINE,2"
00070 CLOSE #1:
00080 OPEN #2:"NAME=TEST.DISP",DISPLAY,INPUT
00090 OPEN #3:"NAME=TEST.INT/SAMPLE,SIZE=512,RECL=21",INTERNAL,OUTPUT
00100 LINPUT #2:A$
00110 PRINT #255:A$
00120 LINPUT #2:A$
00130 PRINT #255:A$
00140 RESTORE #2:
00150 INPUT #2:A$,B
00160 PRINT #255:A$,B
00170 WRITE #3:A$,B
00180 INPUT #2:A$,B
00190 PRINT #255:A$,B
00200 WRITE #3:A$,B
00210 CLOSE #2,FREE:
00220 CLOSE #3:
00230 OPEN #3:"NAME=TEST.INT",INTERNAL,OUTIN
00240 READ #3:
00250 REREAD #3:A$,B
00260 PRINT #255:A$,B
00270 REWRITE #3:A$,B,"NEW"
00280 READ #3:A$,B
00290 PRINT #255:A$,B
00300 RESTORE #3:
00310 READ #3:A$,B,C$
00320 PRINT #255:A$,B,C$
00330 LET COUNT=CNT ! .          MUST SAVE CNT TO PRINT VALUE
00340 PRINT #255:CNT,"REAL VALUE OF CNT IS ";COUNT
00350 PRINT #255:"FILE(3) = ";FILE(3)
00360 PRINT #255:"FILE$(3) = ";FILE$(3)
00370 PRINT #255:"FREESP(3) = ";FREESP(3)
00380 PRINT #255:"WSID$ = ";WSID$
00390 PRINT #255:"REC(3) = ";REC(3)
00400 PRINT #255:"RLN(3) = ";RLN(3)
00410 CLOSE #3,FREE:
```

Sample programs

Program 5 - Printed output

```
SAMPLE 5 - OPEN, CLOSE(FREE), INPUT, LINPUT
          READ, REREAD, WRITE, REWRITE

00040 00050 00060 00070 00080 00090 00100 00110 LINE,1
00120 00130 LINE,2
00140 00150 00160 LINE  1
00170 00180 00190 LINE  2
00200 00210 00220 00230 00240 00250 00260 LINE  1
00270 00280 00290 LINE  2
00300 00310 00320 LINE  1                      NEW
00330 00340 0          REAL VALUE OF CNT IS 3
00350 FILE(3) = 0
00360 FILE$(3) = TEST.INT/SAMPLE/3
00370 FREESP(3) = 509
00380 WSID# = 01
00390 REC(3) = 1
00400 RLN(3) = 21
00410
```

Program 6 - Sample

```
00010 PRINT #255:"SAMPLE 6 - ON ERROR, ERR, LINE, CONTINUE, RETRY"  
00020 TRACE PRINT  
00030 ON ERROR GOTO REPORT  
00040 OPEN #1:"NAME=TEST2/SAMPLE,SIZE=512",DISPLAY,OUTPUT  
00050 PRINT #1:"TEST RECORD"  
00060 CLOSE #1:  
00070 OPEN #1:"NAME=TEST2",DISPLAY,INPUT  
00080 INPUT #1:A  
00090 CLOSE #1,FREE:  
00100 ON ZDIV GOTO FIXZDIV  
00110 LET A=1  
00120 LET B=0  
00130 LET C=A/B  
00140 PRINT #255:C  
00150 STOP  
00160 !  
00170 REPORT: PRINT #255:"ERR=";ERR  
00180 PRINT #255:"LINE=";LINE  
00190 CONTINUE  
00200 !  
00210 FIXZDIV: LET B=1  
00220 RETRY  
00230 END
```

Sample programs

Program 6 - Printed output

```
SAMPLE 6 - ON ERROR, ERR, LINE, CONTINUE, RETRY  
00030 00040 00050 00060 00070 00080 00170 ERR= 726  
00180 LINE= 80  
00190 00090 00100 00110 00120 00130 00210 00220 00130 00140 1  
00150
```

Program 7 - Sample

```
00010 PRINT #255:"SAMPLE 7A" ! INITIALIZE VARIABLES AND ARRAYS THEN CHAIN
00020 OPTION BASE 1
00030 DIM A#*18,C(5)
00040 LET A#="TEST DATA"
00050 LET B=5
00060 DATA 10,20,30,40,50
00070 READ MAT C
00080 OPEN #1:"NAME=CHAIN.TEST/SAMPLE,SIZE=512",DISPLAY,OUTPUT
00090 CHAIN "SAMPLE7B",FILES,A#,B,C
```

```
00010 DIM A#*18,C(5)
00020 USE A#,B,C ! PICK UP CHAINED VALUES AND FILE FROM SAMPLE 7A
00030 PRINT #255:"SAMPLE 7B"
00040 TRACE PRINT
00050 OPTION BASE 1 ! MUST BE SAME AS CHAINED FROM PROGRAM
00060 PRINT #255:A#;B;C(5)
00070 PRINT #255:A#
00080 CLOSE #1,FREE:
00090 PRINT #255:UDIM(C,1)
00100 PRINT #255:MAT C
00110 MAT C=ZER(4)
00120 PRINT #255:MAT C
00130 MAT C=(2*5)*CON(3)
00140 PRINT #255:MAT C
00150 MAT C=C(2)
00160 PRINT #255:MAT C
00170 MAT C=C+C
00180 PRINT #255:MAT C
```


Sample programs

Program 7 - Printed output

```
SAMPLE 7A
SAMPLE 7B
00060 TEST DATA 5 50
00070 TEST DATA
00080 00090 5
00100
  10
  20
  30
  40
  50
00110 00120
  0
  0
  0
  0
00130 00140
  10
  10
  10
00150 00160
  10
  10
00170 00180
  20
  20
```

Program 8 - Sample

```
00010:LOAD SAMPLE8.BUILD ! PROCEDURE TO DRIVE SAMPLE 8
00020:RUN
00030:LINK INDEX
00040:MESSAGES=1
00050:CHOICE=2
00060:MASFILE=TEST.MASTER
00070:KEYSTART=9
00080:KEYLGTH=5
00090:IDXFILE=TEST.INDEX
00100:IDXVOLID=SAMPLE
00110:DUPKEY=N
00120:ENDLINK
00130:LOAD SAMPLE8.TEST
00140:RUN
00150:PRINT #255:"END SAMPLE 8"
```

```
00010 PRINT #255:"SAMPLE8.BUILD"
00020 OPEN #1:"NAME=TEST.MASTER/SAMPLE,SIZE=512,RECL=22",INTERNAL,OUTPUT
00030 WRITE #1,USING FORMK:"RECORD 1","SMITH",12.34
00040 WRITE #1,USING FORMK:"RECORD 2","JONES",56.78
00050 WRITE #1,USING FORMK:"RECORD 3","BURNS",0
00060 CLOSE #1:
00070 FORMK: FORM C 8,C 6,N 8.2
00080 END
```

```
00010 PRINT #255:"SAMPLE8.TEST"
00020 TRACE PRINT
00030 LET A$=PIC$("X") ! INITIALIZE CURRENCY SYMBOL
00040 OPEN #1:"NAME=TEST.MASTER,KFNAME=TEST.INDEX",INTERNAL,INPUT ,KEYED
00050 PRINT #255:KLN(1),KPS(1) ! KEY POSITION , KEY LENGTH
00060 READ #1,USING FORMK,KEY="JONES":A$,NAME$,AMOUNT
00070 FORMR: FORM C 8,C 6,N 8
00080 PRINT #255,USING FORMP:AMOUNT
00090 FORMP: FORM PIC(###.##)
00100 PRINT #255:PIC$("X") ! SET CURRENCY SYMBOL TO "X"
00110 PRINT #255,USING FORMP:AMOUNT
00120 READ #1,KEY="BAKER": EXIT EXIT1
00130 STOP
00140 EXIT1: EXIT NOKEY PRINTERR
00150 STOP
00160 PRINTERR: PRINT #255:"KEY NOT FOUND"
00170 CLOSE #1,FREE: ! FREE TEST FILE
```

Sample programs

Program 8 - Printed output

```
SAMPLE8.BUILD  
SAMPLE8.TEST  
00030 00040 00050 5 9  
00060 00080 $056.78  
00100 X  
00110 X056.78  
00120 00160 KEY NOT FOUND  
00170  
END SAMPLE 8
```

Program 9 - Sample

```
00010:PRINT #255:"SAMPLE 9 - COMMANDS",TAB(1)
00020:CLEAR
00030:10 PRINT #255:"SAMPLE 9 TEST"
00040:20 PRINT #255:"DATE=";DATE$,"TIME=";TIME$
00050:30 PRINT #255:"LINE TO BE DELETED"
00060:40 END 1+83
00070:SAVE SAMPLE9.TESTPROG/SAMPLE,SOURCE
00080:CLEAR
00090:DATE 80/12/04
00100:TIME 15:12:30
00110:RENAME SAMPLE9.TESTPROG,SAMPLE9.TEST2
00120:LOAD SAMPLE9.TEST2
00130:RENUM 100,100
00140:LISTP
00150:210 PRINT #255:"NEW LINE"
00160:DEL 300
00170:REPLACE,SOURCE
00180:LISTP
00190:RUN TRACEP
00200:PRINT #255:CODE
00210:SKIP 1 IF CODE=84
00220:ALERT CODE NOT SET PROPERLY
00230:FREE SAMPLE9.TEST2
00240:PRINT #255:"END SAMPLE 9"
```

Sample programs

Program 9 - Printed output

SAMPLE 9 - COMMANDS

```
00100 PRINT #255:"SAMPLE 9 TEST"
00200 PRINT #255:"DATE=";DATE$,"TIME=";TIME$
00300 PRINT #255:"LINE TO BE DELETED"
00400 END 1+83
00100 PRINT #255:"SAMPLE 9 TEST"
00200 PRINT #255:"DATE=";DATE$,"TIME=";TIME$
00210 PRINT #255:"NEW LINE"
00400 END 1+83
00100 SAMPLE 9 TEST
00200 DATE=80/12/04      TIME=15:12:50
00210 NEW LINE
00400
      84
END SAMPLE 9
```

Program 9 - Display output A

```
PROC SAMPLE9.PROC
PRINT #255:"SAMPLE 9 - COMMANDS",TAB(1)
CLEAR
10 PRINT #255:"SAMPLE 9 TEST"
20 PRINT #255:"DATE=";DATE$,"TIME=";TIME$
30 PRINT #255:"LINE TO BE DELETED"
40 END 1+83
SAVE SAMPLE9.TESTPROG/SAMPLE,SOURCE
CLEAR
DATE 80/12/04
TIME 15:12:30
RENAME SAMPLE9.TESTPROG,SAMPLE9.TEST2
LOAD SAMPLE9.TEST2
00010 PRINT #255:"SAMPLE 9 TEST"
00020 PRINT #255:"DATE=";DATE$,"TIME=";TIME$
00030 PRINT #255:"LINE TO BE DELETED"
00040 END 1+83
RENUM 100,100
LISTP
210 PRINT #255:"NEW LINE"
DEL 300
REPLACE,SOURCE
LISTP
RUN TRACEP
PRINT #255:CODE
SKIP 1 IF CODE=84
FREE SAMPLE9.TEST2
PRINT #255:"END SAMPLE 9"
```

READY INPUT

1000

1.03

1 1

Sample programs

Program 10-Sample

Full-screen processing.

```
00010 PRINT NEWPAGE
00020 LET I=3.14159265
00030 LET A$="A TEST LINE"
00040 LET J=-I
00050 PRINT FIELDS "1,10,N 8,N,N":I
00060 PRINT FIELDS "2,10,N 8,N,N":J
00070 PRINT FIELDS "3,10,N 8.6,N,N":I
00080 PRINT FIELDS "4,10,N 8.5,N,N":J
00090 PRINT FIELDS "5,10,N 11.9,N,N":I
00100 PRINT FIELDS "6,10,N 15.9,N,N":J
00110 PRINT FIELDS "7,10,PIC($$.###),N,N":I
00120 PRINT FIELDS "8,10,PIC($$.###DB),N,N":J
00130 PRINT FIELDS "9,10,PIC(###.#####),N,N":I
00140 PRINT FIELDS "10,10,PIC(###.#####),N,N":J
00150 PRINT FIELDS "11,10,PIC(+++.#####),N,N":I
00160 PRINT FIELDS "12,10,PIC(+++.#####),N,N":J
00170 PRINT FIELDS "13,10,PIC(+*#.#####),N,N":I
00180 PRINT FIELDS "14,10,PIC(+*#.#####),N,N":J
00190 PRINT FIELDS "15,10,PIC(---.#####),N,N":I
00200 PRINT FIELDS "16,10,PIC(---.#####),N,N":J
00210 PRINT FIELDS "17,10,C 11":A$
00220 PRINT FIELDS "18,10,C 11":STR$(I)
00230 PRINT FIELDS "19,10,C 11":STR$(J)
00240 PRINT FIELDS "20,10,V 11":STR$(J)
00250 PRINT FIELDS "21,10,G 11":STR$(J)
00260 PRINT FIELDS "22,10,G 11":J
00270 PRINT FIELDS "23,10,G 11":A$
00280 END
```

Program 10-Display output

Full-screen processing.

```
00050          3
00060         -3
00070        3.141593
00080       -3.14159
00090      3.141592650
00100     -3.141592650
00110    *3.141
00120   *3.141DB
00130  03.14159
00140 -03.14159
00150 +3.1415926
00160 -3.1415926
00170 +*3.1415926
00180 -*3.1415926
00190   3.1415926
00200  -3.1415926
00210  A TEST LINE
00220  3.14159265
00230 -3.14159265
00240 -3.14159265
00250 -3.14159265
00260          -3
00270  A TEST LINE
READY      INPUT                4000                1.01                1 4
```


Sample programs

Program 11-Sample

Full-screen processing.

```
00010 REM FULL SCREEN PROCESSING WITH THE "N X" FORMAT
00020 OPTION BASE 1
00030 DIM B$(16,16)*13,A(16,16)
00040 FOR I=1 TO 16 ! BUILD THE FIELD DEFINITION
00050 FOR J=1 TO 16 ! AND DATA ARRAYS
00060 LET B$(I,J)=STR$(I)&" "&STR$(5*(J-1)+2)&"",N 3,U,N"
00070 LET A(I,J)=16*J+I
00080 NEXT J
00090 NEXT I
00100 MAT B#=B$(256) ! REDIMENSION THE CONTROL ARRAY
00110 PRINT NEWPAGE
00120 PRINT FIELDS MAT B#:MAT A
00130 PRINT FIELDS "22,10,C 50":"THIS USES AN 'N 3' FORMAT"
00140 INPUT FIELDS "22,70,N 1,U,N":I ! WAIT FOR OUTPUT
00150 END
```

Program 11-Display output

Full-screen processing.

17	33	49	65	81	97	113	129	145	161	177	193	209	225	241	257
18	34	50	66	82	98	114	130	146	162	178	194	210	226	242	258
19	35	51	67	83	99	115	131	147	163	179	195	211	227	243	259
20	36	52	68	84	100	116	132	148	164	180	196	212	228	244	260
21	37	53	69	85	101	117	133	149	165	181	197	213	229	245	261
22	38	54	70	86	102	118	134	150	166	182	198	214	230	246	262
23	39	55	71	87	103	119	135	151	167	183	199	215	231	247	263
24	40	56	72	88	104	120	136	152	168	184	200	216	232	248	264
25	41	57	73	89	105	121	137	153	169	185	201	217	233	249	265
26	42	58	74	90	106	122	138	154	170	186	202	218	234	250	266
27	43	59	75	91	107	123	139	155	171	187	203	219	235	251	267
28	44	60	76	92	108	124	140	156	172	188	204	220	236	252	268
29	45	61	77	93	109	125	141	157	173	189	205	221	237	253	269
30	46	62	78	94	110	126	142	158	174	190	206	222	238	254	270
31	47	63	79	95	111	127	143	159	175	191	207	223	239	255	271
32	48	64	80	96	112	128	144	160	176	192	208	224	240	256	272

THIS USES AN 'N 3' FORMAT

RUN	FIELDS	4000	1.01	1 1
-----	--------	------	------	-----

Sample programs

Program 12-Sample

Full-screen processing.

```
00010 REM VARIABLE OPEN OF 4 FILES ON 4 DRIVES
00020 DIM REC#*25
00030 LET J=10
00040 LET K=24
00050 FOR I=1 TO 4 ! OPEN A FILE ON EACH DRIVE
00060 LET FILE#="FILE"&STR$(5-I)&"//"&STR$(I)
00070 REM FILE3 ON DRIVE 2, FILE1 ON DRIVE 4, ETC
00080 OPEN #I:"NAME="&FILE#&","SIZE="&STR$(J*K*80)&","RECL=80",INTERNAL,OUTPUT
00090 NEXT I
00100 LET REC#="THIS IS A TEST RECORD"
00110 FOR J=1 TO 100 ! WRITE 100 RECORDS TO EACH FILE
00120 FOR I=1 TO 4
00130 WRITE #I,USING 140:REC#&" RECORD NUMBER "&STR$(J)&" ON FILE "&STR$(I)
00140 FORM C 80
00150 NEXT I
00160 NEXT J
00170 FOR I=1 TO 4 ! CLOSE THE FILES
00180 CLOSE #I:
00190 NEXT I
00200 END
```

Appendix B. Tables

Tables

1		2	
Hex	Dec	Hex	Dec
0	0	00	0
1	1	10	16
2	2	20	32
3	3	30	48
4	4	40	64
5	5	50	80
6	6	60	96
7	7	70	112
8	8	80	128
9	9	90	144
A	10	A0	160
B	11	B0	176
C	12	C0	192
D	13	D0	208
E	14	E0	224
F	15	F0	240

Table 1. Decimal to hex conversion

Tables

Tables (continued)

	0	1	2	3	4	5	6	7	8	9
0										
1			Note 2					Note 1		
2								!		
3			⌋	⌌	⌍					
4	⌎	⌏		Note 1	⌐				⌑	⌒
5			—							
6			+				â	ä	à	á
7	ã	á	ç	ñ	[.	<	(+	!
8	&	é	ê	ë	è	í	î	ï	ì	β
9] \$ *) ;	^	-	/	Â	Ä			
10	À	Á	Ã	Ä	Ç	Ñ		,	%	_
11	>	?	φ	É	Ê	Ë	È	Í	Î	Ï
12	Ì	˘	:	#	@	'	=	"	φ	a
13	b	c	d	e	f	g	h	i	«	»
14	đ	ý	†	±	·	j	k	l	m	n
15	o	p	q	r	á	ó	æ	,	Æ	☉
16	μ	~	s	t	u	v	w	x	y	z
17	ı	ı	Đ	Ÿ	İ	®	ç	£	¥	₣
18	ƒ	ğ	¶	¼	½	¾	—		—	..
19	ˆ	=	}	A	B	C	D	E	F	G
20	H	I	—	ô	ö	ð	ó	õ	}	J
21	K	L	M	N	O	P	Q	R	ı	û
22	ü	ù	ú	ÿ	\		S	T	U	V
23	W	X	Y	Z	²	Ô	Ö	Ò	Ó	Õ
24	0	1	2	3	4	5	6	7	8	9
25	³	Û	Ü	Ù	Ú					

Notes:

1. Unprintable character
2. Page advance

Use table from left to right

Examples:

decimal code 193 prints A
 decimal code 91 prints \$

The information in this table is used with **CHR\$**

Table 2. Decimal code to print character conversion

The following tables specify the response of the System/23 to any combination of two I/O statements. Statements which are not listed in the table are considered errors.

Preceding I/O statement	Following I/O statement		
	READ	REREAD	RESTORE
READ	Next sequential record	Previous record read	Position to beginning
REREAD	Next sequential record	Previous record read	Position to beginning
RESTORE	First record	Error	Position to beginning

Table 3. OPEN, INTERNAL, SEQUENTIAL, INPUT

Tables

Tables (continued)

Preceding I/O statement	Following I/O statement	
	WRITE	RESTORE
WRITE	Add record at the end	Position to beginning
RESTORE	Record is written as first and only	Position to beginning

Table 4. OPEN, INTERNAL, SEQUENTIAL, OUTPUT

Preceding I/O statement	Following I/O statement**				
	READ	REREAD	REWRITE	RESTORE	DELETE
READ or REREAD	Next sequential record	Read previous record	Update previous READ	Position to beginning	Delete record previously read
DELETE or REWRITE	Read next sequential record	Error	Error	Position to beginning	Error
RESTORE	Read first record	Error	Error	Position to beginning	Error

* Initial position at beginning of file

** Error for diskette files on a WRITE

Table 5. OPEN, INTERNAL, SEQUENTIAL, OUTIN*

Preceding I/O statement	Following I/O statement				
	READ	READ* REC=n ₂	REREAD	RESTORE	RESTORE* REC=n ₂
READ or REREAD	Next sequential record	Read record n ₂	Read previous record	Position to beginning	Position to record n ₂
READ* REC=n ₁	Next sequential record	Read record n ₂	Read previous record	Position to beginning	Position to record n ₂
RESTORE	Read first record	Read record n ₂	Error	Position to beginning	Position to record n ₂
RESTORE* REC=n ₁	Read record n ₁	Read record n ₂	Error	Position to beginning	Position to record n ₂

*NOREC error if record is nonexistent or deleted.

Table 6. OPEN, INTERNAL, RELATIVE, INPUT

The WRITE statement must have a REC=clause. The WRITE must be directed to a deleted record in the file or to record n+1, where n is the last record. If a WRITE REC= is followed by a WRITE REC=, a record is added. If the record specified already exists, a DUPREC error occurs. A RESTORE to a relative file open for output is an error.

Table 7. OPEN, INTERNAL, RELATIVE, OUTPUT

Tables

Tables (continued)

Preceding I/O statement	READ	READ REC= n_2^*	RE-READ	WRITE REC= n_2^{**}	RE-WRITE	RE-WRITE REC= n_2^*	DELETE	DELETE REC= n_2	RE-STORE	RE-STORE REC= n_2^*
READ or REREAD	Read next sequential record	Read record n_2	Read previous record	Add record n_2	Update preceding record read	Update record n_2	Delete previous record read	Delete record n_2	Position to beginning	Position at record n_2
READ REC= n_1	Read next sequential record	Read record n_2	Read record n_1	Add record n_2	Update record n_1	Update record n_2	Delete record n_1	Delete record n_2	Position to beginning	Position at record n_2
Note	Read next sequential record	Read record n_1	Error	Add record n_2	Error	Update record n_2	Error	Delete record n_2	Position to beginning	Position at record n_2
Restore	Read first record	Read record n_2	Error	Add record n_2	Error	Update record n_2	Error	Delete record n_2	Position to beginning	Position at record n_2
RESTORE REC= n_1	Read record n_2	Read record n_2	Error	Add record n_2	Error	Update record n_2	Error	Delete record n_2	Position to beginning	Position at record n_2

Note: This line applies to: WRITE, REC= n_1
 REWRITE
 REWRITE, REC= n_1
 DELETE
 DELETE, REC= n_1

*If a record does not exist, a NOREC error occurs.

**If a record already exists, a DUPREC error occurs.

Table 8. OPEN, INTERNAL, RELATIVE, OUTPUT

Preceding I/O statement	Following I/O statement				
	READ	READ* KEY/ SEARCH	REREAD	RESTORE	RESTORE* KEY/ SEARCH
READ	Read next record by key	Read specified record	Read previous record	Position to first key	Position to specified record
READ KEY= SEARCH=	Read next record by key	Read specified record	Read previous record	Position to first key	Position to specified record
REREAD	Read next record by key	Read specified record	Read previous record	Position to first key	Position to specified record
RESTORE	Read first record by key	Read specified record	Error	Position to first key	Position to specified record
RESTORE KEY= SEARCH=	Read record restored to	Read specified record	Error	Position to first key	Position to specified record

*If there is no KEY to match then a NOKEY error occurs

Table 9. OPEN, INTERNAL, KEYED, INPUT

If A WRITE is followed by a WRITE, a record is added.

Table 10. OPEN, INTERNAL, KEYED, OUTPUT

Tables

Tables (continued)

Preceding I/O statement	Following I/O statement									
	READ	READ* KEY/SEARCH	RE-READ	RE-WRITE	RE-WRITE* KEY=	DE-LETE	RE-STORE	RE-STORE* KEY/SEARCH	DE-LETE* KEY=	WRITE
READ or READ KEY/ SEARCH or REREAD	Read next record by key	R e a d s p e c i f i e d r e c o r d	Read previous record	Update previous record read	U p d a t e s p e c i f i e d r e c o r d	Delete previous record read	P o s i t i o n t o f i r s t k e y	P o s i t i o n t o s p e c i f i e d k e y	D e l e t e s p e c i f i e d r e c o r d	A d d r e c o r d b y k e y
REWRITE or REWRITE KEY= or DELETE or DELETE KEY= or WRITE	Read next record by key		Error	Error	Error	Error				
RESTORE	Read first record by key		Error	Error	Error					
RESTORE KEY/ SEARCH n ₂	Read record n ₂		Error	Error	Error					

*If there is no key to match, then a NOKEY error occurs

Table 11. OPEN, INTERNAL, KEYED, OUTIN

Preceding I/O statement	Following I/O statement		
	INPUT	LINPUT	RESTORE
INPUT	Next logical line	Next logical line	Position to beginning
LINPUT	Next logical line	Next logical line	Position to beginning
RESTORE	First logical line	First logical line	Position to beginning

Table 12. OPEN, DISPLAY, INPUT

Preceding I/O statement	Following I/O statement	
	PRINT	RESTORE
PRINT	Next logical line written	Position to beginning
RESTORE	First logical line written	Position to beginning

Table 13. OPEN, DISPLAY, OUTPUT

Appendix C. Performance tips and techniques

Introduction

This appendix identifies areas that significantly affect program and system performance. Performance is enhanced if it is initially designed carefully and thoughtfully.

Performance tips and techniques

BASIC statements and functions

A functionally enriched BASIC language on System/23 makes it possible to achieve the same results with various combinations of BASIC statements. Read carefully the complete set of options of each BASIC statement and, in particular, to the specially provided set of 45 system functions. Using system functions can eliminate many lines of program statements, thereby improving processing time. Performance can also be enhanced by selecting and using the BASIC statements carefully.

This section contains general comments on the use of BASIC to assist you in selecting combinations of statements and functions. Many of the following comments and examples will become noticeably significant when processed repetitively in loops or collectively with other statements.

Statement length (255 characters)

Combine statements where possible to take advantage of the 255-character maximum statement length.

Example

Do:

```
10 PRINT USING 20: "TOTALS", A, B, C
20 FORM X 9,C 6,SKIP 1,3*N 8.2
```

Instead of:

```
10 PRINT USING 20: "TOTALS"
20 FORM X 9,C 6
30 PRINT USING 40: A, B, C
40 FORM 3*N 8.2
```

Constants in variables

Maintain constants in a variable if they are used repeatedly to initialize other variables. This executes somewhat faster than assignment from a constant and uses less storage.

Example

Do:

```
20 LET I1=1
•
•
300 LET COUNTER=I1
•
•
500 LET SWITCH=I1
```

Instead of:

```
•
•
•
300 LET COUNTER=1
•
•
500 LET SWITCH=1
```


Performance tips and techniques

BASIC statements and functions (continued)

Array initialization with MAT assignment

Use the MAT assignment statement to initialize an array, since it is nearly as fast as the simple assignment.

Example

Do:

```
10 DIM ARRAY (10)
20 MAT ARRAY=(10)
```

Instead of:

```
10 DIM ARRAY (10)
20 FOR I=1 TO 10
30 ARRAY(I)=0
40 NEXT I
```

Simple subscripts

Performance can be improved by not specifying an expression as an array subscript.

Example

Do:

```
X=N+1
ARRAY(X)=5
```

Instead of:

```
ARRAY (N+1)=5
```

Arithmetic guidelines

Consider the arithmetic guidelines in the following examples.

Examples

Do:

```
250 LET B=A*.5
500 LET B=A+A
650 LET B=A*A*A
```

Instead of:

```
250 LET B=A/2
500 LET B=A*2
650 LET B=A**3
```

Expressions—avoid repetitive evaluation

Avoid repetitive evaluation of the same expression in a statement. Evaluate the expression once, and save the result in a variable for use in subsequent statements.

Example

Do:

```
300 LET A=C*3+D
310 LET X=A+B
320 LET Y=A+E
```

Instead of:

```
310 LET X=C*3+D+B
320 LET Y=C*3+D+E
```

Performance tips and techniques

BASIC statements and functions (continued)

System functions

System/23 has 45 preprogrammed system functions. Each of these functions has a specific purpose and was designed to make program development easier and more efficient. Use the system functions whenever possible because they execute faster than the same capability written in the BASIC language.

The following is a list of some useful system functions and a brief description of their purpose.

Function Name		Purpose
POS	Position	Locates a string of characters within a second string of characters
SRCH	Search	Searches a one-dimensional array for the location of a specific character string or numeric value
SREP\$	String replace	Replaces a substring of characters with a new substring of characters within a larger character string
RPAD\$	Pad blanks right	Adds blanks to the right of a character string
LPAD\$	Pad blanks left	Adds blanks to the left of a character string

RTRM\$	Trim blanks right	Removes blanks from the right of a character string (The V FORM statement specification removes trailing blanks from a character value being input.)
LTRM\$	Trim blanks left	Removes blanks from the left of a character string

You should become familiar with each of the system functions. Refer to the main section of this manual for a complete description of each individual system function.

Performance tips and techniques

Logic control

The sequential execution of a BASIC program can be modified by many of the BASIC statements. Use the following general guidelines when selecting the method of branching for each step of a program.

Labels for branching

Use labels as targets for branching statements to improve the readability of a program.

Branching to labels executes as fast as branching to line numbers. Also, storage is saved if the label is referenced from more than one location in the program.

Example

```
500 GOTO BEGIN
```

Subroutine use

Use subroutines to handle commonly used portions of a program. Invoke these subroutines using either the GOSUB/RETURN statements or the DEF/FNEND define function statements. The GOSUB/RETURN combination executes faster than the function call. However, the performance of a subroutine is highly dependent on the BASIC code within the subroutine. Defined functions have specific characteristics that are not available to the GOSUB/RETURN combination. Review the features of the DEF/FNEND and the GOSUB/RETURN statements in the main section of this manual before selecting the subroutine technique to use in a program.

IF statement capabilities

You can improve performance by taking full advantage of the capabilities of the IF statement. By using the logical AND/OR comparison and the THEN/ELSE clause capability of the IF statement, you can often avoid the need for more IF statements and additional code in a program.

Example

Do:

```
200 IF A=B AND C=D THEN Z=A ELSE Z=B
```

Instead of:

```
200 IF A=B THEN GOTO 210
205 GOTO 215
210 IF C=D THEN GOTO 225
215 LET Z=B
220 GOTO 230
225 LET Z=A
230 ...
```

IF statement ordering

Place the most frequently occurring IF condition at the beginning of a series of IF statements. Example 1:

A customer order to be read from a data-entry file consists of different record types and numerous individual transactions.

Performance tips and techniques

Logic control (continued)

Record type	Identification
A	Header
B	Customer name and address
C	Transaction
•	•
•	•
C	Transaction
D	Trailer

Do:

```
100 IF RECTYPE$="C" THEN GOTO 3000
110 IF RECTYPE$="A" THEN GOTO 1000
120 IF RECTYPE$="B" THEN GOTO 2000
130 IF RECTYPE$="D" THEN GOTO 4000
```

Instead of:

```
100 IF RECTYPE$="A" THEN GOTO 1000
110 IF RECTYPE$="B" THEN GOTO 2000
120 IF RECTYPE$="C" THEN GOTO 3000
130 IF RECTYPE$="D" THEN GOTO 4000
```

Moving the test for record C to the beginning of the list would result in 1800 fewer IF statements executed, assuming 100 groups with an average of 10 transactions per group.

Example 2:

Cascading the IF statements may also improve performance and reduce storage requirements.

Do:

```
200 IF A ><1 THEN GOTO NOT1
210 IF B=1 THEN X=0
220 IF B=2 THEN X=1
230 IF B=3 THEN X=2
240 GOTO CONTIN
250 NOT1: IF B=1 THEN X=3
260 IF B=2 THEN X=4
270 IF B=3 THEN X=5
280 CONTIN: ...
```

Instead of:

```
200 IF A=1 AND B=1 THEN X=0
210 IF A=1 AND B=2 THEN X=1
220 IF A=1 AND B=3 THEN X=2
230 IF A><1 AND B=1 THEN X=3
240 IF A><1 AND B=2 THEN X=4
250 IF A><1 AND B=3 THEN X=5
```


Performance tips and techniques

Logic control (continued)

Loop design

In general, design loops using FOR/NEXT statements, instead of the IF/GOTO combination of statements.

Example

Do:

```
200 FOR I=1 TO 10
  .
  .
  .
300 NEXT I
```

Instead of:

```
200 I=I+1
210 IF I=11 THEN GOTO 310
  .
  .
300 GOTO 200
```

Tight loops

Always inspect loops for unnecessary code; whenever possible, the code should be removed from the loop. This includes nonexecutable statements such as FORM (see "Nonexecutable statements" in this appendix).

Example

Do:

```
10 LET A=B+1
20 FOR X=1 TO 1000
030 IF D(X)<A THEN D(X)=A
040 NEXT X
```

Instead of:

```
10 FOR X=1 TO 1000
20 LET A=B+1
30 IF D(X)<A THEN D(X)=A
40 NEXT X
```

Coding techniques

Nonexecutable statements

Some required BASIC statements are considered nonexecutable since they do not functionally alter the program during execution. As these statements are encountered during the execution of a program, the system requires a small amount of time to identify the instruction and proceed to the next sequential statement. The nonexecutable statements are DIM, DEF/FNEND, FORM, EXIT, and REM. Whenever possible, move these nonexecutable statements out of loops.

Example

Do:

```
200 DIM A(10)
210 FORM C 5, N 3
220 FOR I=1 to 10
  .
  .
  .
300 NEXT I
```

Instead of:

```
200 FOR I=1 to 10
  .
  .
  250 DIM A(10)
  260 FORM C 5, N 3
  .
300 NEXT I
```

Performance tips and techniques

Coding techniques (continued)

Remarks

Comments (REM statements) are considered nonexecutable. The system requires a small amount of time to identify a REM statement. Place comments on individual instructions using an exclamation point instead of REM statements whenever possible. Placing comments on individual statements improves performance and also saves storage by eliminating the need for a line number and the REM statement.

Example

Do:

```
100 FOR A=1 to 15 ! Process items
  •
  •
400 NEXT A
```

Instead of:

```
100 FOR A=1 to 15
110 REM Process items
  •
  •
490 NEXT A
```

Array redimensioning

The redimensioning of an array saves storage. When you originally dimension an array to a predetermined maximum dimension and find that in the course of running the program you don't need the maximum size, you can redimension downward.

Example:

Do:

```
100 DIM A$(100)*20
110 INPUT A ! Operator types in a number
           of items to be input
120 FOR I = 1 to A
130 INPUT A$(I)
140 NEXT I
.
.
.
190 MAT A$ = A$(A) ! Redimension down to
           the number of items input
200 CHAIN "NEXTONE", A$, A
```

Instead of:

```
100 DIM A$(100)*20
110 INPUT A ! Operator types in a number
           of items to be input
120 FOR I = 1 to A
130 INPUT A$(I)
140 NEXT I
.
.
.
190 REM without redimensioning
           you will pass a 100 element array
200 CHAIN "NEXTONE", A$, A
```

Performance tips and techniques

I/O techniques

Buffer space considerations

System/23 does not necessarily do a physical I/O operation each time a READ or WRITE statement is executed. If enough storage is available, System/23 may put up to 7.5K bytes of data per file into a buffer before actually reading from or writing to the diskette or disk. By taking advantage of this buffering, heavily I/O bound programs can be made to run faster.

Points to remember are:

- Sequential vs. relative access of files
 - Sequential access files can be buffered up to 7.5K bytes of data. For relative access files, System/23 tries to keep as many 512 byte buffers as needed to hold one logical record.
 - Use sequential access whenever possible.
- Priority of accessing files
 - When more than one file will be accessed in a program, access the most frequently used file first to ensure that as much buffer space as possible is assigned to this file.
- Need for closing files
 - CLOSE files that are no longer needed. This frees space for buffers.
- Using CHAIN statement
 - Additional space for buffers can be provided by breaking the program into a number of programs by using the CHAIN statement. However, be sure that the benefits are not offset by the length of time it takes to execute the chain.
- Selecting record length

-
- System/23 format diskettes and the 5247 Disk have a block length of 512 bytes. Records that cross sector boundaries require additional physical I/O. To avoid this, choose a record length which, when incremented by one, divides evenly into 512. (for example: 63, 127, 511). The extra byte is the control byte that System/23 attaches to each record in type 04 files.

OPEN statement considerations

An OPEN statement requires many physical I/O operations and is time consuming. The time required can be reduced in the following ways:

- Restrict use of CLOSE and OPEN statements
 - Close a file only if it will not be used again or if storage is a problem. This avoids unnecessary OPEN statements.
- Position most-used files first
 - Position frequently opened files first in the diskette directory. This can be done by creating these files first on empty diskettes before adding additional files. Directory position does not affect the performance of OPEN statements in files on the 5247 Disk.
- Specify drive numbers
 - Specify drive numbers and VOLIDs in the OPEN statement whenever possible.

Performance tips and techniques

I/O techniques (continued)

- Group files
 - If a BASIC program has many files located on the 5247 Disk, group the files in one volume and specify the VOLID in the OPEN statement.

Access time reduction for keyed files

The time required to access keyed files can be reduced in the following ways:

- Use keyed, sequential access
 - Whenever possible, use keyed, sequential access.
- Specify KW= in the OPEN statement
 - For large key files, specify "KW=" in the OPEN statement (see "OPEN statement").
- Regenerate key files after update
 - After adding records to the master file, regenerate key files using the Create Index File Customer Support Function. This places all keys in sorted order within the key file.

General I/O performance guidelines

- Allow sufficient file size
 - System/23 files will be automatically extended when full, but this will slow down any file access. For better performance, whenever possible, create a file as large as will be required.
- Compress files
 - Compress multi-extent files (see "DIR command"). Do this by copying the file to an empty diskette or a different disk volume using the Copy Customer Support Function.

-
- Multiple rewrites to the same record, see “Programming considerations” under “REWRITE statement.”
 - Copy diskettes with media errors.
 - If a diskette begins to have media errors (see “DIR command”), transfer all data to another diskette. Do this using the Copy or Recover Diskette Customer Support Functions.
 - Place files for best access
 - When a program will be accessing more than one file, the files should be on separate diskettes or be positioned close together when on a single diskette.
 - Place 5247 Disk files on the same volume.
 - Select best file format
 - Use Basic and H-exchange files only to transfer data between System/23 and other systems.
 - Always use System/23 format files for normal processing.
 - BASIC language considerations
 - Unformatted I/O is faster than formatted I/O.
 - MAT I/O is faster than scalar processing.

Performance tips and techniques

I/O techniques (continued)

Selection of data file access method

Choosing the proper access method for your data files is one of the more important decisions you must make.

Whether to use the *sequential*, *relative*, or *keyed* access method depends on your application.

- Accessing individual records
 - *Relative access.* A fast method of accessing an individual record is directly by means of the relative record number of the desired record. For example, in an inventory file, it is possible to convert the item number into a record number. Item numbers could be 1 to 1000. Item number 52 would be record 52 in the file.
 - *Keyed-access indexing.* This is the next fastest method to access individual records. A pointer to the master file data record is maintained in an index file. This is the most commonly used access method because existing keys such as item numbers can be used.
 - *Sequential access.* Processing a file sequentially to find an individual record is time consuming because the file must be read from the beginning until the proper record is found.

-
- Processing sequential files
 - If an *entire file* is to be processed from beginning to end, sequential access is the fastest method.
 - The fastest method to process a file sequentially is to sort the master file into the desired order before processing.
 - To process a RELATIVE file sequentially, starting at a specified record number, first open the file RELATIVE, and then execute a RESTORE #X, REC=A statement. Subsequent READ statements without a REC= clause will read each record sequentially from the file.
 - Processing keyed files
 - If a file is to be processed SEQUENTIAL in some cases and RELATIVE in others, it may be more appropriate to create an index (key) file. The system can then access the master file (1) SEQUENTIAL by accessing the index file or (2) RELATIVE by providing a key to the index file.

Performance tips and techniques

I/O techniques (continued)

Main storage index area for keyed access method

Access to a master file record using an index (key) file can be improved substantially if you maintain an index area in main storage that points to the index (key) file. To do this, use the KW= parameter, which is included in the OPEN statement.

Example

```
30 OPEN #1:"NAME=TAXES,KFNAME=TAXKEY,KW=50",  
INTERNAL,INPUT,KEYED
```

In the preceding statement, 50 bytes of main storage have been allocated for index file pointers.

Performance can be improved for random key access to a file when an optimum KW parameter is assigned. Refer to the "OPEN statement" section of this manual for a complete description of the KW parameter and instructions about how to calculate the optimum KW value.

Index file sorting

Many applications, such as inventory, make use of an index file with pointers that allow fast access to desired records. If the index file is sorted, access to a master record will be faster than if the index file is not sorted. The index file for a master file is automatically placed in sorted order when it is initially created by the Create Index File Customer Support Function. (see *Customer Support Functions*, Volume II).

As new items are added to the master file, the item number key (item number is specified as the key) is added to the end of the index file. Depending on the activity of adding and deleting records, the index file should be periodically *recreated* so that the new index record is placed in its proper location and the unwanted index records are deleted.

Index

A

- about this book iii
 - prerequisites iii
 - related publications iv
- ABS(X) 6
- absolute value
 - See ABS(X)
- access method 498
- access time reduction keyed files 496
- accuracy 13
- action tables, I/O 231
- add or delete a volume 77
- addition and multiplication 19
- AIDX and DIDX 7
 - example (AIDX) 8
 - example (DIDX) 9
 - programming considerations 9
 - syntax 7
- ALERT command 10
 - syntax 10
- allocation of printer 312
- allocation, disk 78
- allocation, diskette 78
- alphabetic character set
 - See character set
- alphabetic characters 48
- AND (relational expression) 214
- arc tangent
 - See ATN(X)
- arithmetic arrays 12
 - BASE 0 12
 - BASE 1 12
 - declared 12
 - dimensioned 12
 - one-dimensional 12
 - two-dimensional 12
- arithmetic constant 17
 - example 17
- arithmetic data 13
 - accuracy 13
 - arithmetic constant 17
 - arithmetic data formats 14
 - blink 192
 - fixed-point format 15
 - floating-point format 16
 - integer format 14
 - magnitude 13
 - selecting an arithmetic format 17
 - significance 13
- arithmetic data formats 14
 - fixed-point format 15
 - examples 15
 - floating-point format 16
 - example 16
 - integer format 14
 - examples 14
 - selecting an arithmetic format 17
 - example 17
- arithmetic expressions 23
- arithmetic expressions and operations 18
 - example 18
 - operators 19
 - rules 19
 - syntax 18
- arithmetic format 17
- arithmetic guidelines 483
- arithmetic hierarchy 23
 - example 23
 - order of priority 23
- arithmetic variables 25
 - examples 25
- array element references 134
- array expressions
 - See MAT assignment
- array initialization 482
- array name, character 428
- ARRAY redimensioning 276, 492
- arrays 12, 26, 43
 - data-item 26
 - elements 26
 - example 27
 - maximum subscript 29
 - OPTION BASE 0 26
 - OPTION BASE 1 26
- arrays, arithmetic
 - See arithmetic arrays
- arrays, character
 - See character arrays

Index

arrays, declaring
 See declaring arrays
arrays, redimensioning
 See redimensioning arrays
ascending index 7
assignment 271
assignment statements
 See LET statement
asynchronous communications terminal 76
asynchronous problem information 76
ATN(X) 30
attention and inquiry 31
 Cmd/Attn 31
 Inq Key 32
attributes, display 190
attributes, Full-screen 189
AUTO command 33
 examples 34
 increment 33
 line-num 33
 programming considerations 34
 syntax 33

B

backup and restore 77
BASE 0 12
BASE 1 12
BASIC statements 35
 CHAIN 40
 CLOSE 62
 CONTINUE 67
 DATA 79, 80
 DEF 83
 DELETE 95
 DIM 103
 END 124
 EXIT 127
 FNEND 83
 FOR 153
 FORM 158
 GOSUB 202
 GOTO 207
 IF,THEN,ELSE 211
 INPUT 218
 LET 255
 LINPUT 261
 MAT 270
 NEXT 153
 ON 284
 OPEN 289
 OPTION 305
 PAUSE 310
 PRINT 314
 RANDOMIZE 339
 READ 340
 RELEASE 353
 REM 354
 REREAD 361
 RESERVE 363
 RESTORE 368
 RETRY 372
 RETURN 202, 373
 REWRITE 374
 STOP 396
 TRACE 424
 USE 426
 WRITE 433
 5217 Printer 321
BASIC statements and functions 480
batch data transfer 75
binary synchronous problem information 76
 binary synchronous problem information 76
blanks 38
 rules 38
bsc online test 76
buffer space considerations 494
built in functions 418
byte 38

C

- catenation
 - See concatenation
- CEIL(X) 38
 - example 38
- ceiling
 - See CEIL(X)
- CHAIN statement 40
 - data-item 40
 - example 41
 - FILES 40
 - pgmname 40
 - programming considerations 42
 - syntax 40
- changing BASIC statements 123
- changing line numbers 123
- character array name 428
- character arrays 43
 - example 43
- character constants 44
 - examples 44
- character data 44
- character expressions 45
 - examples 45
 - related subjects 46
 - syntax 45
- character position (POS) 163
- character set 48
 - alphabetic characters 48
 - characters not displayable 54
 - EBCDIC characters 51
 - graphic characters 50
 - hexadecimal representation 51
 - numeric characters 48
 - special characters 49
- character string 59
- character string input (LINPUT) 261
- character variable 428
- character variables 58
 - example 58
- CHR\$(X) 59
 - example 59
- CLEAR command 60
 - parameters 60
 - syntax 60
- CLEAR DATA 60
- clear display screen
 - See PRINT statement
- CLOSE statement 62
 - example 63
 - parameter 62
 - programming considerations 63
 - syntax 62
- closing a procedure file 330
- CMD/ATTN 31
- CMDKEY 64
- CNT 64
- CODE 65
- coding techniques 491
- collating sequence, change 73
- commands
 - See SYSTEM commands
- comments
 - See remarks
- communications 69
- CON 66
- concatenation 66
 - example 66
- constants 17, 44
- constants in variables 481
 - examples 481
- CONTINUE statement 67
- control attributes 193
- control reserve 356
- conventional representation 14
- Copy 72
 - copy 72
- COS(X) 68
- create index file 73
- cross reference 68
- cursor position 193
- Customer Support Functions 69
 - asynchronous communications terminal 76
 - asynchronous problem information 76
 - batch data transfer 75
 - bsc online test 76
 - change collating sequence 73
 - copy 72

Index

- create index file 73
- disk customer support functions 77
 - add or delete a volume 77
 - backup and restore 77
 - prepare disk 77
 - recover disk 77
- display diskette labels 72
- list files 74
- list storage information 75
- load machine update 71
- prepare batch data transfer 75
- prepare diskette 71
- prepare sort 74
- recover diskette 73
- replace 73
- select machine update 71
- set up asynchronous communications 75
- set up binary synchronous communications 75
- sort 74

D

- data 12, 44, 80
- data buffering 78
- data display 118
- data file 233
- data file access method 498
- DATA files 79
 - create a display file 79
 - edit an existing DISPLAY file 79
 - example 79
- data formats 14
- data items 26
- DATA statement 80
 - example 81
 - parameter 80
 - programming considerations 81
 - syntax 80
- data transfer, batch 75
- DATE command 82
 - syntax 82
- DATES\$ 82
- decimal
 - See FORM statement
- declaring arrays 82
- DEF, FNEND statement 83
 - multiple-line function 85
 - example 87
 - syntax 85
 - one-line function 83
 - example 84
 - syntax 83
 - programming considerations 89
 - use of functions 88
- DEL command 93
 - example 94
 - parameter 93
 - programming considerations 94
 - syntax 93
- DELETE statement 95
 - example 96
 - parameter 95
 - programming considerations 96
 - syntax 95
- deleting a file
 - See FREE command
- descending index 7
- descending index (MAT assignment)
 - See also AIDX and DIDX
 - drive number 97
- device address parameter 97
- device sharing 98, 100
- DIDX 7, 102
- DIDX (array name) 102
- digit specifiers 176
- DIM statement 103
 - declaring arrays 104
 - example 104
 - example 103, 106
 - parameter 103
 - programming considerations 107
 - syntax 103
- dimension, upper (UDIM) 425
- dimensioning arrays
 - See DIM statement
- DIR command 108
 - disk 113
 - drive number 113
 - example 115

- parameter 113
- syntax 113
- diskette 108
 - example 110
 - parameter 108
 - syntax 108
- diskettes, using DIR without a 5247 Disk Unit 108
- disk
 - disk considerations 148
 - file types 145
 - prepare 77
 - recover 77
- disk access state (VOLID) 430
- disk storage requirements 145
- disk,saving on 359
 - REPLACE 359
 - SAVE 385
- diskette access state (VOLID) 430
- diskette storage requirements 145
 - file types 145
- diskette,saving on 359
 - REPLACE 359
 - SAVE 385
- DISPLAY 118
 - example 118
 - files and data 118
- display diskette labels 72
 - display labels 72
- display files 118
- DISPLY(X) 120
- division 21
- dollar sign (PIC\$) 311
- DROP/FREE command 120
 - example 121
 - parameter 121
 - syntax 121
- DUPREC 128
- dynamic file extension 122

E

- EBCDIC characters 51
- edit DISPLAY file 79
- editing a program or data file 123
 - adding statements 123
 - changing line numbers 123
 - deleting statements 123
 - programming considerations 124
 - replacing statements 123
- element 26
- ELSE
 - See IF, THEN, ELSE statement
- end of file (EXIT) 128
- END statement 124
 - examples 125
 - parameter 124
 - programming considerations 125
 - syntax 124
- ERR 125
- error handling 126
- evaluation expressions 23
- executing a procedure file 329
- execution order
 - See order of execution
- EXIT statement 127
 - error conditions 127
 - CONV 127
 - DUPREC 128
 - EOF 128
 - IOERR 128
 - NOKEY 128
 - NOREC 128
 - PAGEFLOW 129
 - example 130
 - parameter 127
 - programming consideration 131
 - syntax 127
- EXP(X) 133
- exponent specifiers 182
 - example 182
- exponential
 - See EXP(X)
- exponentiation 21

Index

expressions 134, 483
expressions, arithmetic
 See arithmetic expressions and operations
expressions, array
 See MAT assignment
expressions, character
 See character expressions
expressions, relational
 See IF, THEN, ELSE statement

F

FIELDS

 See full-screen processing

file 328

file extension, dynamic 122

file name 147

file reference parameter 137

 example 137

file searches 138

file sharing 140

 controlled 141

 default 141

 programming considerations 143

 regulated 140

 restrictions 140

 summary 142

file size 145

file space (FREESP) 188

file specification parameter 147

 control attributes 189

 display attributes 192

 control attributes 193

 example 193

 examples 195

 highlight 193

 invisible 192

 normal 192

 programming considerations 196

 underline 193

 examples 147

 parameters 190

 programming considerations 196

 syntax 189

5247 Disk considerations 148

file types 151

FILE(N) 150

FILE\$(N) 150

file-spec 147

FILENUM 136

 example 136

files 79, 252

files, display 118

files, internal 224

files, key-indexed 224

files, related subjects 149

files, internal I/O 224

fixed-point format 152

floating currency symbol 152

floating-point formula 16

FNEND statement 152

FOR and NEXT statements 153

 examples 155

 FOR syntax 153

 NEXT syntax 154

 programming considerations 156

FORM statement 158

 output positions 158

 digit specifiers 176

 display character data 158

 example of V format 171

 examples of L format 169

 examples of N format 166

 examples of PD format 168

 examples of V format 170

 examples of ZD format 168

 exponent specifier 182

 I/O list items 159

 increase the spacing 159

 insertion characters 180

 PIC specifications 172

 programming considerations 185

 reading same values 160

 syntax 161

 trailing characters 183

 written into a record 160

 ZD parameter 167

FORMAT 346

format, arithmetic 17

formatting I/O files 187
FREE command 187
FREESP(N) 188
full screen processing 189
function names 427, 428
function references 134
functions 69
functions, built in 418
functions, defined
 See DEF, FNEND statement

G

GO command 199
 examples 201
 parameters 199
 programming considerations 201
 syntax 199
GOSUB and RETURN statement 202
 examples 204
 parameter 202
 programming considerations 204
 syntax 202
GOTO statement 207
 computed GOTO syntax 207
 example 208
 parameters 207
 programming considerations 209
 simple GOTO syntax 207
graphic characters 50
guidelines 483

H

HELP STATUS command 209
 example 209
 syntax 209
HEX\$(A\$) 210
hexadecimal representation 51
hierarchy, arithmetic
 See arithmetic hierarchy
HOLD 210
HX (H exchange) 145

I

I/O action tables 231, 469
I/O files 225
I/O list items 159
I/O performance guidelines 496
I/O techniques 495
IF statement capabilities 487
IF statement ordering 487
IF, THEN, ELSE statement 211
 BASIC statements 212
 example 213
 logical operators, expressions 214
 parameter 211
 programming considerations 216
 relational expression 213
 relational functions and definitions 213
 syntax 211
increment loop (NEXT) 153
index file sorting 501
index file, create 73
index keys 217
indexing, BASE 0 or 1 305
INPUT FIELDS 189
INPUT statement 218
 example 219
 parameter 218
 programming considerations 221
 syntax 218
inq key 32
inquiry 31
inquiry key
 See attention and inquiry key
integer format
 See arithmetic data
internal constants 224
internal files
 See key-indexed files
 See relative record files
internal I/O file formatting 225
 formatted (with USING) 225
 unformatted (without USING) 226
internal I/O files 227

Index

internal machine fixes (LINK) 259
internal representation of characters
 See character set
internal variables
 See system variables
interrupt 228
interrupt handling 229
 priority 229
intrinsic functions
 See system functions
introduction 1
inverted print 305
IOERR 128
ITN(X) 230

K

key description legend and tables 234
 key description tables 239
 legend 234
key starting position (KPS) 253
key work area size 302
key-indexed files 252
 created 252
 process 252
keyboard 231
 programming consideration 232
keyboard-generated data files 233
 changed 233
 created 233
 length 233
 listed 233
keyed files 496
keys
 See keyboard
KLN(N) 253
KPS(N) 253
KSTAT\$ 254

L

L format 170
labels 254
 example 254
labels for branching 486
leading and trailing attributes 191
LEN(A\$) 254
length of a character (LEN) 254
length of key for file (KLN) 253
length of record (RLN) 379
LET statement 255
 example 256
 parameter 255
 programming considerations 255
 syntax 255
LINE 257
line control
 See PRINT statement
line function
 See DEF, FNEND statement
line numbers 258
line reference 258
 example 258
LINK command 259
 example 260
 parameter 259
 syntax 259
LINPUT statement 261
 example 262
 parameter 261
 syntax 261
list diskette or disk information 74
list files 74
list items, I/O 159
list storage information 75
LIST, LISTP command 263
 examples 266
 parameter 263
 programming considerations 264
 syntax 263
LOAD command 267
 example 267
 parameter 267

- programming considerations 268
- syntax 267
- load machine update 71
- LOG(X) 269
- logarithm
 - See LOG
- logic control 486
- logical expressions
 - See IF, THEN, ELSE statement
- logical operators, expressions 214
- loop (FOR and NEXT) 153
- loop design 490
- LPAD\$(C\$, X) 269
- LTRM\$(C\$) 270

M

- magnitude 13, 270
 - See also arithmetic data
- main storage index area 500
- MAT AIDX 7
- MAT assignment
 - See AIDX and DIDX
- MAT assignment (addition, subtraction, scalar multiplication) 271
 - example (addition and subtraction) 271
 - example (scalar multiplication) 273
 - parameter 271
 - programming considerations 271
 - syntax 271
- MAT assignment (scalar value) 274
 - example 275
 - parameter 274
 - programming considerations 274
 - syntax 274
- MAT assignment (simple) 276
 - ARRAY redimensioning 276
 - example 276
 - parameter 276
 - programming considerations 277
 - syntax 276
- matrix
 - See arrays

- matrix operations
 - See AIDX and DIDX
- MAX (X1, X2, X3, ...) 278
- maximum subscript 29
- maximum value 278
 - See also arithmetic data
- MERGE command 279
 - example 281
 - parameter 279
 - programming considerations 280
 - syntax 279
- MIN (X1, X2, X3, ...) 281
- minimum value
 - See arithmetic data
- modifying a procedure file 329
- modifying a screen 189, 263
- multiple line function
 - See DEF, FNEND statement

N

- N format 167
- names, variable
 - See variable names
- negative 22
- NEWPAGE 315, 319
- next
 - See FOR and NEXT statement
- NEXT statement 154
- NOKEY 128
- non-katakana 388
- NONE (GOSUB, GOTO) 203
- nonexecutable statements 491
- NOREC 128
- numbers 258
- numeric array name 427
- numeric characters 48
- numeric data formats
 - See arithmetic data
 - See FORM statement, (N specification)
- numeric to string conversion (STR\$) 401
- numeric variable 427

Index

O

- offline diskette files 283
 - diskette files ,offline 283
- OFLOW 284
- ON GOSUB 202
- ON GOTO 372
- ON statement 284
 - example 287
 - parameter 284
 - programming considerations 287
 - syntax 284
- online test 76
- OPEN statement 289
 - OPEN DISPLAY (syntax) 290
 - example 294
 - parameter 291
 - syntax 291
 - OPEN INTERNAL syntax 295
 - example 300
 - OPEN parameter table 304
 - parameter 296
 - programming considerations 300
 - syntax 296
 - parameter 289
- OPEN statement considerations 495
- operations,arithmetic 18
- operators 134
- OPTION BASE 0 26
- OPTION BASE 1 26
- OPTION statement 305
 - parameter 305
 - programming considerations 307
 - syntax 305
- OR 214
- ORD(A\$) 308
- order of execution 307
 - rules 307
- ordinal value 308
- output positions 158
- overstruck characters
 - See FORM statement

P

- packed decimal
 - See FORM statement (PD specification)
- PAD
 - See LPAD\$
- page overflow 309
- parameter, device address
 - See device address parameter
- parameter, file reference
 - See file reference parameter
- parameter, file specification
 - See file specification parameter
- PAUSE statement 310
- PD format 169
- performance tips and techniques 479
- PI
 - See internal constants
- PIC 172
- PIC\$ 311
- POS 163
- POS(A\$,B\$,X) 311
- positive/negative operations 22
- prepare batch data transfer 75
- prepare disk 77
- prepare diskette 71
 - prepare 71
- prepare sort 74
- prerequisites iii
- PRINT BELL
 - See PRINT statement
- print control
 - See printer assignment
- print data list delimiters
 - See PRINT statement
- PRINT FIELDS 189
- PRINT statement 314
 - parameter 314
 - programming considerations 316
 - syntax 314
- USING 315
 - 5217 printer statements 321
- print zones
 - See PRINT statement

- printer assignment 312
- printer spacing 320
- PROC command 327
 - parameter 327
 - syntax 327
- procedure 328, 443
- procedure file 328
 - closing a procedure file 330
 - creating a procedure file 328
 - data 331
 - example 332
 - executing a procedure file 329
 - interrupting 331
 - modifying a procedure file 329
 - nesting a procedure file 331
 - storage 332
 - using the ALERT command 331
 - using the SKIP command 331
- PROCEED command 333
 - examples 334
 - parameter 333
 - programming considerations 335
 - syntax 333
- processing 189
- PROCIN 335
- program 443
- program or data file 123
- PROTECT command 336
 - examples 338
 - parameter 336
 - syntax 336

R

- random numbers
 - See RANDOMIZE statement
- RANDOMIZE statement 339
- READ statement 340
 - example 341
 - parameter 340
 - programming considerations 342
 - syntax 340
- READ statement (with file reference) 344
 - example 346

- key length 348
- parameter 344
- programming considerations 348
- syntax 344

- REC(N) 349
- record I/O files 349
- record length (RLN) 379
- record number (REC) 349
- recover disk 77
- recover diskette 73
 - recover 73
- redimensioning arrays 350
 - example 351
 - rules 350
- reference parameter 137
- reference, line 258
- referencing, substrings 351
- related subjects, files 149
- relational expressions 135, 351
- relational function 213
- relative record file 351
 - accessed by 352
- RELEASE statement 353
 - example 353
 - parameter 353
 - programming considerations 353
 - syntax 353
- REM statement 354
 - example 355
- remarks 355
 - example 355
- RENAME command 356
 - example 356
 - syntax 356
- RENUM command 357
 - parameter 357
 - programming considerations 358
 - syntax 357
- replace 73
- REPLACE command 358
 - parameter 359
 - programming considerations 360
 - syntax 359
- replacing a statement 123, 361
- representation of characters 51

Index

REREAD statement 361
 example 363
 parameter 361
 syntax 361
RESERVE statement 363
 parameter 363
 programming considerations 364
 syntax 363
reserved words 366
RESTORE statement 368
 example 368
 parameter 368
 syntax 368
RESTORE statement (with file reference) 369
 parameter 369
 programming considerations 371
 syntax 369
RETRY statement 372
 example 372
 parameter 372
 programming considerations 373
RETURN statement 373
REWRITE statement 374
 example 376
 parameter 374
 programming considerations 376
 syntax 374
RLN(N) 379
RND(X) 379
ROUND(X,M) 380
RPAD\$(A\$,X) 380
RPT\$(A\$,M) 381
RTRM\$(A\$,) 381
RUN command 382
 example 383
 parameter 382
 programming considerations 383
 syntax 382

S

sample procedure or sample program
 See Appendix A
sample program 443
SAVE command 385
 examples 386
 parameter 385
 programming considerations 386
 syntax 385
scalar multiplication (MAT assignment)
 See MAT assignment
screen 393
screen,clear (NEWPAGE) 315, 319
search
 See file searches
 See SRCH
select machine update 71
selection of data file access method 498
set up asynchronous communications 75
set up binary synchronous communications 75
SGN(X) 387
sharing 140
 See also file sharing
sharing files (types of file-sharing) 117
sharing the 5246 Diskette Unit 98
sharing the 5247 Disk Unit 100
SHIFT(X) 388
sign of a number
 See arithmetic data
significance 13
 See also arithmetic data
simple subscripts 482
SIN(X) 389
size of array (DIM) 103
SKIP 164, 331
SKIP command 390
 example 391
 syntax 390
SKIP lines
 See FORM statement
sort 74
SORT command 392
 parameter 392

- syntax 392
- sorting 501
- space considerations 494
- space, work area (HELP STATUS) 209
- space,file (DROP/FREE) 120
- spaces
 - See blanks
- spacing, FORM 158
- special character set
 - See character set
- specification parameter 147
- split screen 393
- SQR(X) 393
- square roots
 - See SQR(X)
- SRCH (array,X), SRCH (array\$,X) 394
- SREP\$(A\$,M,B\$,C\$) 394
- standard format
 - See integer format
- statement length
 - See BASIC statements
- statement numbers
 - See line numbers
- statements
 - See BASIC statements
- statements, adding 123
- status line
 - See character set
 - See device sharing
 - See DISPLAY
- STOP statement 396
 - example 396
 - syntax 396
- storage use 332, 397
 - example 399
- STR\$(X) 401
- string 59, 403
- string to numeric conversion 427
- subjects related to arithmetic expressions 22
- subjects related to character expressions 46
- SUBPROC command 402
 - exceptions 402
 - parameter 402
 - syntax 402
- subroutine use 486

- subroutines
 - See GOSUB statement
 - See RETURN statement
- subscripted variables
 - See arrays
- substring referencing 403
 - example rule 5 407
 - example using subscripts 407
 - examples of substring referencing 405
 - extraction of characters 405
 - programming consideration 408
 - replacement of characters 406
 - rules 404
- subtraction 21
- support 69
- syntax description 410
- SYSTEM command 414
- SYSTEM commands 415
 - ALERT 10
 - AUTO 33
 - CLEAR 60
 - DATE 82
 - DEL 93
 - DIR 108
 - DROP 120
 - FREE 120
 - GO 199
 - HELP STATUS 209
 - LINK 259
 - LIST 263
 - LISTP 263
 - LOAD 267
 - MERGE 279
 - PROC 327
 - PROCERR 333
 - PROTECT 336
 - REMANE 356
 - RENUM 357
 - REPLACE 359
 - RUN 382
 - SAVE 385
 - SKIP 331, 390
 - SORT 392
 - SUBPROC 402
 - SYSTEM command 414

Index

TIME 423
VOLID 430
system functions 418, 484
system keywords
 See reserve words
 See system commands
system variables 421
 CMDKEY 421
 CNT 421
 CODE 421
 DATE\$ 421
 ERR 421
 FILENUM 421
 KSTAT\$ 421
 LINE 421
 PROCI 421
 TIME\$ 421
 WSID\$ 421

T

TAB function
 See PRINT statement
tables 469
 See also arrays
TAN(X) 422
techniques 479, 491
TEST 32
THEN
 See IF, THEN, ELSE statement
tight loops 490
TIME command 423
 syntax 423
TIME\$ 423
tips and techniques 423, 479
TRACE statement 424
 syntax 424
trailing characters 183
 examples 184
trim
 See LTRM\$
 See RTRM\$

U

UDIM (array,X) 425
UFLOW 285
unformatted READ 348
use of blanks 38
USE statement 426
 data-item 426
 syntax 426
USING 225, 315, 344, 361, 374, 433

V

V format 170
VAL(A\$) 427
variable names 427
 character variable or character array name 428
 function names 428
 numeric variable or numeric array name 427
variables 25, 58, 134
variables, arithmetic 429
variables, character 429
variables, internal 429
VOLID command 430
 example 432
 parameters 430
 syntax 430
volume identification (DIR) 108

W

whole numbers (CEIL) 38
work area
 See storage use
WRITE statement 433
 parameter 434
 programming considerations 435
 syntax 433
write-protection (PROTECT) 336
WSID\$ 436
 programming consideration 438

X

XREF

See LIST, LISTP command

Z

ZD format 168

ZDIV 285

ZER and CON 439

CON syntax 441

example 442

parameter 441

ZER syntax 439

example 440

parameter 439

5

5247 Disk customer support functions 77

add or delete a volume 77

backup and restore 77

prepare disk 77

recover disk 77



READER'S COMMENT FORM

SA34-0109-2

BASIC Language Reference

Your comments assist us in improving the usefulness of our publications; they are an important part of the input used in preparing updates to the publications. IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Please do not use this form for technical questions about the system or for requests for additional publications; this only delays the response. Instead, direct your inquiries or requests to your IBM representative or the IBM branch office serving your locality.

Corrections or clarifications needed:

Page	Comment
------	---------

Cut or Fold Along Line

Please indicate your name and address in the space below if you wish a reply.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A.
(Elsewhere, an IBM office or representative will be happy to forward your comments.)

Reader's Comment Form

Cut Along Line

Fold and tape

Please Do Not Staple

Fold and tape

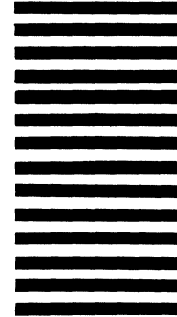


NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK
POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Information Development, Dept 27T
P.O. Box 1328
Boca Raton, Florida 33432



Fold and tape

Please Do Not Staple

Fold and tape



SA34-0109-2
Printed in U.S.A.

READER'S COMMENT FORM

SA34-0109-2

BASIC Language Reference

Your comments assist us in improving the usefulness of our publications; they are an important part of the input used in preparing updates to the publications. IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Please do not use this form for technical questions about the system or for requests for additional publications; this only delays the response. Instead, direct your inquiries or requests to your IBM representative or the IBM branch office serving your locality.

Corrections or clarifications needed:

Page	Comment
------	---------

Cut or Fold Along Line

Please indicate your name and address in the space below if you wish a reply.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A.
(Elsewhere, an IBM office or representative will be happy to forward your comments.)

Reader's Comment Form

Cut Along Line

Fold and tape

Please Do Not Staple

Fold and tape



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Information Development, Dept 27T
P.O. Box 1328
Boca Raton, Florida 33432



Fold and tape

Please Do Not Staple

Fold and tape





This Newsletter No. SN34-0828
Date September 22, 1982
Base Publication No. SA34-0109-2
Previous Newsletters None

**IBM System/23
BASIC Language Reference**

© IBM Corp. 1981, 1982

This Technical Newsletter provides replacement pages for the subject publication. Pages to be inserted and/or removed are:

393, 394

A technical change to the text or to an illustration is indicated by a vertical line to the left of the change.

Summary of Amendments

- Additional information for using the BASIC SRCH array function is provided.

Note. Please file this cover letter at the back of the manual to provide a record of changes.



System/23

BASIC Language Reference

© IBM Corp. 1981, 1982

This Technical Newsletter provides replacement pages for the subject publication. Pages to be inserted and/or removed are:

67, 68
333 through 336
357 through 366
385 through 390
407 through 410
513, 514

A technical change to the text or to an illustration is indicated by a vertical line to the left of the change.

Summary of Amendments

1. Clarification of the LOCK portion of the REPLACE and SAVE commands.
2. Revised information on ROS level 1.05.
3. Revised pages of the book index.

Note: Please file this cover letter at the back of the manual to provide a record of changes.





International Business Machines Corporation

SA34-0109-2
Printed in U.S.A.