SMA STANDARDS

SMA/BASIC Language Specification

SMA: 101

April 1986

SPECTRUM MANUFACTURERS ASSOCIATION

sma

-- NOTICE --

SMA standards are designed to serve the public interest through eliminating misunderstandings between manufacturers and purchasers, facilitating interchangeability and improvement of products, and assisting the purchaser in selecting and obtaining, with minimum delay, the proper product for his particular need. Existence of such standards shall not in any respect preclude any member or non-member of SMA from manufacturing or selling products not conforming to such standards, nor shall the existence of such standards preclude their voluntary use by those other than SMA members whether the standard is to be used either domestically or internationally.

Some material contained herein is designated as proprietary by Pick Systems. Any use of such material other than in connection with the use or operation of Pick-based software is not authorized.

Published by
SPECTRUM MANUFACTURERS ASSOCIATION
9740 Appaloosa Rd., Suite 104
San Diego, CA 92131

Foreword: This document provides a set of syntax definitions for the SMA/BASIC language. This language, provided by all Spectrum Manufacturers Association member systems, is the primary tool for defining algorithmic processes in the systems. It provides means for accessing the file structures, for accepting input information from the operator or other data systems, and for preparing printed reports as well as operator CRT screens. The language is called SMA/BASIC, having been derived from the original BASIC programming language. Many extensions for handling the data structures of the SMA system have been added. This document is intended to serve as a guide to the preparation of programs that can be moved from one SMA system to another. For the details on any specific system, the user should refer to the manufacturer's reference manual.

The SMA Executive Board wishes to thank the following individuals and organizations for their contributions in the preparation of this document:

F. Kacerek, Ultimate Corp.
T. Holland, Pick Systems
R. Whitaker, Pertec Computer Co.
M. Hannigan, Applied Digital Data Systems, Inc.
R. Burns, CDI Information Systems, Inc.
C. Wilson, General Automation, Inc.

CONTENTS

1.0 Scope

1.1 Implementation Objectives: It is the objective of this document to provide the user with a defined set of the language to enable the preparation of programs that can be moved from one system to another with minimum difficulty.

1.2 Inclusions: This document includes all the commonly available statements with syntactical representations to clearly define the usage permitted with each. The common set of intrinsic functions are also documented in the same context. Although most of the statements and functions are provided by all the SMA systems, some implementations may restrict or limit the use of some due to the configurations of the hardware or other factors.

1.3 Exclusions: In this version of this document, the syntactical aspects of the language are addressed. There are related issues of the treatment of certain statements by the "run time" support in various systems that will be addressed by future versions of this document.

2.0 Definitions:

2.1 Nomenclature: Within this document capitalized words represent tokens within the language and must be included as shown. The use of parentheses (()) is explicit in the language and must be considered part of the statement or function. The use of double quotes (") or single quotes (') is also explicit in the language and must be considered part of the expression.

The use of braces ({}) indicates that the included field or string is optional. The use of ellipsis (...) means that right-trailing recursion is acceptable for the foregoing bracketed field or string. The slash (/) is used to separate items in a list, one of which must be chosen.

2.2 Abbreviations: The following symbolic identifiers are used in the syntactical definitions throughout the following sections.

call.arg	List of variables, expressions or dimensioned variables that are passed to a Subroutine.
char.exp	Expression that evaluates into a character.
cmnt	Comment.
cond.exp	Conditional expression that evaluates to zero (FALSE) or non-zero (TRUE).
cond.stmnt	Conditional statement followed by then.else.clause (refer to section 4.5).
conv.exp	Refer to section on Conversion Strings (2.10).
dict.exp	Expression that evaluates to a dictionary name, the character string DICT or a null character string.
dyn.arry	Dynamic array.
exp	Expression.
file.exp	Expression that evaluates to a file name.
file.var	Variable name assigned when a file is opened.
format.str	Refer to section on Format Strings (2.9).
hd.exp	Refer to section on Heading/Footing Strings (2.11).
int	Integer value (constant, variable or expression).

2.2 Abbreviations (continued):

int.exp	Expression that evaluates into an integer.
item.exp	Expression that evaluates to an item identifier.
mat.var	Dimensioned variable (also called a matrix).
name	Name of an external subroutine or program.
num	Numeric value (constant, variable or expression).
num.cnst	Constant consisting of the digits 0,1,2,...9 with optional prefix plus or minus sign and an optional decimal point. Maximum of 14 digits, evaluates to numeric.
num.exp	Expression that evaluates to a numeric result.
num.label	Numeric label.
print.list	List of expressions, each with optional format strings, separated by commas which imply tabulation.
sel.var	Variable to which a select list is assigned.
stmnt	A simple or compound statement on a single line.
stmnt.cmpnd	A compound statement consists of two or more simple statements seperated by semi-colons.
stmnts	One or more statements, on multiple lines.
str	String value (constant, variable or expression).
str.cnst	Constant consisting of zero or more printable characters enclosed in matching single quotes (') or double quotes ("). Evaluates to a character string.
str.exp	Expression that evaluates into a character string.
sub.arg	List of local variables or dimensioned variables in a subroutine whose values are passed from a calling routine.
var	Variable.
var.name	Name which is assigned as a pseudonym for a value or another variable in an equate statement (4.4).

2.3 Arithmetic Operators:

+	Arithmetic addition (also unary plus)
-	Arithmetic subtraction (and unary minus)
/	Arithmetic division, quotient
*	Arithmetic multiplication
^	Arithmetic exponential

2.4 Logical Operators:

AND or &	Logical AND operation
OR or !	Logical OR operation

2.5 Relational Comparison Operators:

LT or <	Less than comparison
GT or >	Greater than comparison
LE or <=	Less than or equal comparison
NE or #	Not equal comparison
GE or >=	Greater than or equal comparison
EQ or =	Equal comparison
MATCH{ES}	Pattern matching test

2.6 String Operator:

CAT or :	String concatenation

2.7 Post-fix String Extract Operations:

(Follows a variable reference.)

[int.exp,int.exp]	Substring Extraction
<int.exp{,int.exp{,int.exp}}>	Dynamic array extraction

2.8 Operator Precedence:

The various operators are considered to have an order of precedence for evaluation when an expression is not explicitly ordered by the use of parentheses. Evaluation begins with those variables coupled by operators with the highest rank and proceeds to those with lower rank. Evaluation within an expression of a set of operations that have the same precedence rank proceeds from left to right. Expressions inside parentheses are evaluated before operations outside of the parentheses.

Operator		Precedence	
	Function Evaluation	0	highest
[]	Substring Extraction	0	
<>	Dynamic Array Extraction	0	
^	Exponential	1	
*	Multiplication	2	
/	Division	2	
+	Addition	3	
-	Subtraction	3	
	Format Mask	4	
:	Concatenation	5	
	Relational Comparisons	6	
&	Logical AND	7	
!	Logical OR	7	lowest

2.9 Format String:

The format string provides special control information for the formatting operation performed upon data specified in a format expression. The value of a format string has the following general form:

{j}{n{m}}{Z}{,}{c}{$}{(format.mask)}

j Specifies justification. May specify R for right justification or L for left justification. The default justification is left.

n Single numeric digit defining the number of digits to print out following the decimal point. If n = 0, the decimal point will not be output following the value.

m Scaling factor specified by a single numeric digit which 'descales' the converted number by the 'mth' power of 10. Because SMA/BASIC assumes 4 decimal places (unless otherwise specified by a PRECISION statement), to descale a number by 10, m should be set to 5; to descale a number by 100, m should be set to 6; etc.

Z Parameter specifying the suppression of value zero.

, Parameter for output which inserts commas between every thousands position of the value. (European versions may insert decimals rather than commas.)

c The following five symbols are credit indicators which are parameters of the form:

C Causes the letters CR to follow negative values and causes two blanks to follow positive or zero values.

D Causes the letters DB to follow positive values; two blanks to follow negative or zero values.

M Causes a minus sign to follow negative values; a blank to follow positive or zero values.

2.9 Format String (continued):

E Causes negative values to be enclosed within angle brackets (<value>); a blank follows positive or zero values.

N Causes the minus sign of negative values to be suppressed.

$ Parameter for output which appends a dollar sign to the leftmost position of the value, prior to conversion. The printed symbol for $ may differ depending on the country of use.

format.mask Parameter enclosed in optional parentheses with values as follows:

#n specifies that the data is to be filled on a field of 'n' blanks.

*n specifies that the data is to be filled on a field of 'n' asterisks.

%n specifies that the data is to be filled on a field of 'n' zeros and to force leading zeros into a fixed field.

NOTE: Any other character, including parentheses may be used as a field fill. Mixed mode fields may be formed by repeating the control characters (#, *, and %).

2.10 Conversion String:

The conversion functions (see section 5.4) use a character string to specify the type of conversion. The conversion is made assuming conversion codes from the following set:

D	Convert date to internal format.
G	Extract group of characters.
L	Test string length.
MC	Mask characters by numeric, alpha, or upper/lower case.
ML	Mask left-justifies decimal data.
MR	Mask right-justified decimal data.
MT	Convert time to internal format.
MX	Convert ASCII to hexadecimal.
P	Test pattern match.
R	Test numeric range.
T	Convert by table translation. The table file and translation criteria must be given.

2.11 Heading/Footing String:

These strings are used to specify headings and footings for page orientated output. Note that 'hd.options' must be surrounded by single quotes, double quotes are not allowed in this context.

{str.exp}{{'hd.options'}{str.exp}}...

where hd.options are one or more of the following:

C	Center text on the line.
D	Current date.
L	Carriage return and line feed.
P{n}	Current page number right justified in field of n blanks. If n is not specified, it is assumed to be 4.
T	Current time and date.

Note that any string enclosed in single quote marks is considered as a heading option declaration. To present a single quote within the printed heading, two quotes must be used to represent it.

2.12 Pattern Matching String:

The pattern matching string is used to specify the control information for the pattern matching expression. The value of the string consists of one or more of the following:

"string" or 'string'	Literal string test.
nN	Numeric String Test, n digits.
nA	Alphabetic string test, n characters.
nX	Any characters test, n characters.

where: n may be zero which implies any number of characters, including none.

3.0 Expressions:

3.1 Definition:

An expression may be any constant, variable, string, function, expression enclosed within parentheses, or a compound expression. A compound expression is formed by combining two or more expressions with appropriate operators (eg. "exp op exp").

3.2 Arithmetic Expressions:

Arithmetic expressions are formed by using arithmetic operators to combine expressions that evaluate to a numeric result.

3.3 Relational Expressions:

Relational expressions are formed by applying a relational operator to a pair of arithmetic or string expressions. A relational expression always evaluates to 1 if the relation is true, and to zero if the relation is false.

3.4 Logical Expressions:

Logical expressions are formed by applying a logical operator to a pair of conditional expressions (cond.exp) and evaluates to 1 (TRUE) or zero (FALSE).

3.5 String Expressions:

String expressions are formed by applying string operators to expressions. The resulting value is a string.

3.6 Format Expression:

Format expressions are formed by combining two expressions with no intervening operator. The value of the left expression will be formatted according to the rules specified in the right expression. The value of the expression on the right is called the format string (refer to section 2.9).

3.7 Pattern Matching Expression:

Pattern matching expressions are a form of relational expression where the operator is either MATCH or MATCHES. The string value on the left of the operator is analyzed according to the pattern matching string value on the right. If the left value matches the pattern specified, the resulting expression value is 1 (TRUE), otherwise the resulting value is zero (FALSE) (refer to section 2.12).

4.0 Statement Syntax Definitions:

4.1 Definition:

The structure of a statement includes an optional label field and a statement body. The end of a statement is marked by the end of the line or a semicolon (;). The label field is separated from the statement body by one or more spaces. The structure may be expressed as follows:

```
{num.label} statement.body
```

4.2 Comment Statement:

```
*   {cmnt}
```

4.3 Variable Replacement Statements:

```
var = exp

MAT mat.var = exp

MAT mat.var = MAT mat.var

dyn.arry<int.exp{,int.exp{,int.exp}}> = str.exp
```

4.4 Variable Setting and Structuring Statements:

```
CLEAR

DATA exp{,exp}...

EQU{ATE} var.name TO num/str/CHAR()/var/mat.var(int.exp)
    {, var.name TO num/str/CHAR()/var/mat.var(int.exp)}...

COM{MON} var/mat.var{(int{,int})}
    {,var/mat.var{(int{,int})}}...

DIM mat.var(int{,int}){,mat.var(int{,int})}...

PRECISION int
 where int is a number from 0 to 6 inclusive.
```

4.5 Then.else.clause Structures:

Many SMA/BASIC statements provide the ability to perform different actions based on some condition. These statements use the then.else.clause to direct the flow of control.

Single line forms:

```
cond.stmnt THEN stmnt {ELSE stmnt}

cond.stmnt ELSE stmnt
```

Multi-line THEN, single line ELSE form:

```
cond.stmnt THEN
     stmnts
     ...
END {ELSE stmnt}
```

Single line THEN, multi-line ELSE form:

```
cond.stmnt THEN stmnt {ELSE
     stmnts
     ...
END}
```

Multi-line THEN, multi-line ELSE form:

```
cond.stmnt THEN
     stmnts
     ...
END {ELSE
     stmnts
     ...
END}
```

Multi-line ELSE form:

```
cond.stmnt ELSE
     stmnts
     ...
END
```

4.6 Program Flow Control Statements:

```
GO{TO} num.label

ON int.exp GO{TO} num.label{,num.label}...

IF cond.exp then.else.clause

BEGIN CASE
   {cmnt...}
     CASE cond.exp
          stmnts
          ...
     {CASE cond.exp
          stmnts
          ...}
     ...
END CASE

FOR var = num.exp TO
       num.exp {STEP num.exp} {WHILE/UNTIL cond.exp}
     {stmnts}
     ...
NEXT var

LOOP {stmnt} WHILE/UNTIL cond.exp DO {stmnt} REPEAT

LOOP
     {stmnts}
     ...
WHILE/UNTIL cond.exp DO
     {stmnts}
     ...
REPEAT
```

4.7 Subroutine Control Statements:

GOSUB num.label

ON int.exp GOSUB num.label{,num.label}...

CALL name/@var {(call.arg{,call.arg}...)}

SUBROUTINE name {(subroutine.arg{,subroutine.arg}...)}

RETURN

RETURN TO num.label
Note that this form of RETURN may only be used with internal subroutines called via a GOSUB and not from calls made via the CALL statement.

EXECUTE str.exp
Note that the str.exp is treated as a TCL statement, that a DATA statement passes 'input' to an EXECUTE statement, that a select list is returned to the executing program, and that up to and including 5 levels of EXECUTE may be used.

4.8 File Access Statements:

```
OPEN {dict.exp,} file.exp TO file.var then.else.clause

READ var FROM file.var,item.exp then.else.clause

READU var FROM file.var,item.exp {LOCKED stmnt}
     then.else.clause
            where LOCKED option specifies an action to take if
            the appropriate file group is already locked.

READV var FROM file.var,item.exp,int.exp then.else.clause

READVU var FROM file.var,item.exp,int.exp {LOCKED stmnt}
     then.else.clause
            where LOCKED option specifies an action to take if
            the appropriate file group is already locked.

MATREAD mat.var FROM file.var,item.exp then.else.clause

MATREADU mat.var FROM file.var,item.exp {LOCKED stmnt}
     then.else.clause
            where LOCKED option specifies an action to take if
            the appropriate file group is already locked.

WRITE{U} exp ON file.var,item.exp

WRITEV{U} exp ON file.var,item.exp,int.exp

MATWRITE{U} mat.var ON file.var,item.exp

DELETE file.var,item.exp

SELECT {file.var/exp} {TO sel.var}

READNEXT var {,var} {FROM sel.var} then.else.clause

CLEARFILE file.var

RELEASE {file.var,item.exp}
```

4.9 Tape (Removable Media) Statements:

READT var then.else.clause

WRITET exp then.else.clause

REWIND then.else.clause

WEOF then.else.clause

4.10 Multiuser File and Execution Lock Statements:

Note that at least 64 unique locks are provided (numbered from 1 through 64).

LOCK int.exp {ELSE stmnt}

UNLOCK int.exp

4.11 Terminal Input/Output and Printer Output Statements:

PROMPT char.exp

INPUT var{,int.exp}{:}

BREAK ON/OFF/exp
where: exp = 0, Break key is off
0, Break key is on
Note that for every break-off that is issued, a corresponding break-on must be issued.

ECHO ON/OFF/exp
where: exp = 0, Echo is off
0, Echo is on

HEADING hd.exp

FOOTING hd.exp

PAGE {exp}

PRINTER ON/OFF/CLOSE

PRINT {ON int.exp} {print.list {:}}

CRT {print.list {:}}

4.12 Program Termination Statements:

STOP {exp{,exp}...}

ABORT {exp{,exp}...}

CHAIN str.exp
SMA recommends that the 'I' option not be used in conjunction with the RUN verb for reliability, transportability, and data integrity reasons.

ENTER name/@var
Note that the program that is to be entered from an ENTER statement must be cataloged.

4.13 Compiler Directives:

INCLUDE {file.exp} item.name

NULL

END

4.14 Miscellaneous Statements:

SLEEP exp
where exp is either:
a numeric value which specifies the number of seconds to sleep; or,
a string value which specifies a sleep until the specified time of the form HH:MM{:SS}.

5.0 Intrinsic Functions:

5.1 Arithmetic and Logical Functions:

ABS(num.exp)

INT(num.exp)

NOT(cond.exp)

NUM(exp)

SQRT(num.exp)

RND(num.exp)

COS(num.exp)

SIN(num.exp)

TAN(num.exp)

LN(num.exp)

EXP(num.exp)

PWR(num.exp,num.exp)
where the first num.exp is raised to the power value denoted by the second num.exp.

REM(num.exp,num.exp)
where first num.exp is the numerator and second num.exp is the denominator.

5.2 Character String Functions:

ASCII(str.exp)

EBCDIC(str.exp)

CHAR(num.exp)

SEQ(char.exp)

SPACE(num.exp)

5.2 Character String Functions (continued):

```
STR(str.exp,num.exp)

TRIM(str.exp)

LEN(str.exp)

COUNT(str.exp,char.exp)

DCOUNT(str.exp,char.exp)

FIELD(str.exp,str.exp,num.exp)

COL1()
COL2()

INDEX(str.exp,str.exp,num.exp)

@(int.exp,int.exp) or @(int.exp)
     where:  -1= Clear screen
             -2= Home
             -3= Clear to end of screen
             -4= Clear to end of line
```

5.3 Dynamic Array Manipulation Functions:

```
INSERT(dyn.arry,int.exp,int.exp,int.exp,str.exp)

DELETE(dyn.arry,int.exp,int.exp,int.exp)

EXTRACT(dyn.arry,int.exp,int.exp,int.exp)
 or
dyn.arry<int.exp{,int.exp{,int.exp}}>

REPLACE(dyn.arry,int.exp,int.exp,int.exp,str.exp)
 or
dyn.arry<int.exp{,int.exp{,int.exp}}> = str.exp

LOCATE str.exp IN dyn.arry{<int.exp{,int.exp}>},num.exp
 BY str.exp SETTING var then.else.clause
```

5.4 Miscellaneous Functions:

TIME()

DATE()

TIMEDATE()

SYSTEM(int)
where int = 0 - Returns error code
1 - Printer ON/OFF
2 - Page Size
3 - Page Depth
4 - Lines Remaining
5 - LIne Counter
6 - Page Number
7 - Terminal Type
8 - Tape Record Length

ICONV(exp,conv.exp)
where the value of the second expression is the conversion string (see section 2.10) and specifies the type of input conversion to be applied to the string value resulting from the first expression.

OCONV(exp,conv.exp)
where the value of the second expression is the conversion string (see section 2.10) and specifies the type of output conversion to be applied to the string value resulting from the first expression.