✓ pp6 4
6/14/88

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 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.

N/A

SMA STANDARD

Magnetic Media Interchange Specification

SMA: 201

March 1987

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.

Some material contained herein is designated as proprietary by individual member companies of SMA listed below. Any unauthorized use of such proprietary information is prohibited.

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

SMA:201
MAGNETIC MEDIA INTERCHANGE STANDARD - DRAFT 1.8

Foreword: The purpose of this document is to establish a standard for interchangeable magnetic media within the Spectrum Manufacturers Association community. The goal of the standard is to assure continuing data compatibility between different SMA implementations which have physically compatible media. Additionally, it is meant to serve as a description of SMA magnetic media protocols so that other operating systems may exchange data with an SMA system through the off-line tape and diskette mechanisms.

Within this document, it becomes evident that inconsistencies and incompatibilities exist on both hardware and software levels. In some of these cases, the standard outlines restrictive guidelines which should be followed to maximize portability. The Technical Committee of SMA is agressively pursuing these areas of deficiency, with the intent of establishing and publishing standards which will improve the situation in these areas of the media interchange.

Change Notice: Draft 1.5 contains changes made to sections 4 through 7 in regards to 1/4" tape cartridge standards. These changes were approved by the SMA Technical Standards Committee on August 6, 1986.

Draft 1.7 contains changes made to wording within the changes made as Draft 1.5. These changes were approved by the SMA Technical Standards Committee on October 2, 1986. Other changes were made to sections 3.2, 4.4 (inserted), 4.11 and 4.15.

Draft 1.8 contains a warning added to section 4.16.

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

D. Credicott,	Nixdorf Computer Software Co.
I. Sandler,	CIE Systems, Inc.
J. Gallant,	Prime Computer, Inc.
D. Harman,	Systems Management, Inc.
C. Saunders,	Fujitsu Microsystems Of America, Inc.
C. Wilson,	General Automation, Inc.
T. Steforos,	Altos Computer, Inc.

CONTENTS

CONTENTS

1.0 Introduction

1.1 Inclusions: On an SMA system, data can be produced in several ways, which affect the format of the actual data on the magnetic media. This standard covers the two most common mechanisms: (1) the TCL "T-DUMP" verb, which is used to dump file items to a tape; and (2) direct input and output from programs written in the SMA/BASIC language.

Additionally, the standard includes a list of tape handling TCL verbs, with a brief functional description of each one.

1.2 Exclusions: Excluded from this version of the standard are the two other techniques of creating data tapes under an SMA system. These include: (1) the TCL "SAVE" verbs; and (2) the commands which deal with spooler input and output on tape. These two areas will be covered in future revisions of the standard.

1.3 Philosophy: Because of the wide variety of removable media utilized on the SMA systems, complete interchange and maximum functionality cannot always be achieved. As a hypothetical example, one class of device may support a physical blocksize of up to 32000 bytes, whereas another may support a maximum of 8000 bytes. In this case, the standard would be compelled to adopt a blocksize upper limit of 8000, in order to assure that tapes produced by either system could be processed by the other.

The restrictions, limitations, and recommended operating procedures set forth in this document have been selected by a "majority rules" philosophy. That is, when more than one choice exists within an area that requires a unique standard, the selection was determined on the basis of maximizing benefit to the largest number of users and manufacturers. In many cases, the resulting standard is a "lowest common denominator" within the community, and may be exceeded by several of the implementations.

It must be emphasized that nothing within this document was intended to be, or should be construed as, a favorable or unfavorable comment on the equipment, software, or expertise of any vendor.

1.4 Changeability: For a number of reasons, this standard must be considered as a dynamic document, subject to revision and extension. First of all, there are technical areas upon which a standard has not yet been agreed, as outlined in the "Exclusions" paragraph. Secondly, the content of the standard is heavily influenced by the hardware and software capabilities of the various manufacturers, both of which are constantly changing. Thirdly, there is substantial diversity among the tape and diskette devices regarding the handling of certain conditions, such as end-of-volume detection. Establishing a common standard in areas such as these may require operating system modifications across all vendors.

1.5 Restrictions: Because of the wide variety of removable media utilized on the SMA systems, complete interchange is limited. This limitation is most pronounced in the ways which the hardware deals with the end-of-volume situation. As a consequence, only single-volume media can be processed with any reliability across different vendor equipment. The long term solution for this deficiency will most likely require a common software circumvention within the operating system itself.

In the meantime, to maximize transportability, media should be constructed in such a fashion that no data file spans across more than one volume. If the amount of data physically requires multiple volumes, then programmer action should be taken to subdivide the data into groups such that each group can be contained on a separate volume as an integral file.

2.0 Definitions

2.1 Scope: This chapter provides definitions of the terms, phrases, and notational abbreviations that are used within this document. Some of these definitions carry a broader scope in the context of this standard than they usually imply, and are therefore included.

2.2 Terms and Phrases:

Attachment size: When a magnetic media device is attached (made available) to a process, the logical blocksize is either specified in the command or implied by a default value. The value remains constant until it is explicitly changed or until the device is detached from the process. All data blocks (excluding any label records) will be written at this logical blocksize. Whenever possible, the physical blocksize should be the same as the logical blocksize.

2.2 Terms and Phrases (continued):

EOF: Acronym for "end of file".

Interchangeable magnetic media: This term refers to a data recording capability, utilizing magnetic technology, in which the component actually storing the data can easily be dismounted from one computer system and mounted on another. Within this standard, the term refers specifically to tape and floppy diskette facilities.

System delimiters: SMA systems utilize certain hexadecimal characters as delimiters within the file system. The name, acronym, hexadecimal value, and usage are summarized below:

Segment Mark	SM	X'FF'	Delimits items (records)
Attribute Mark	AM	X'FE'	Delimits attributes (fields) within an item (record)
Value Mark	VM	X'FD'	Delimits multiple values within an attribute (field)
Sub-Value Mark	SVM	X'FC'	Delimits multiple sub-values within a value
Buffer Mark	BM	X'FB'	Buffer control

Tape: Within the context of this standard, the term "tape" refers to any of the commercially available magnetic media which is interchangeable. Specifically, it includes not only conventional 1/2" reel-to-reel tape facilities, but also includes 1/4" cartridge tape and floppy diskette technology.

TCL: Acronym for "terminal control language".

2.3 Abbreviations:

Within this standard, the following abbreviations are used in describing the syntax of tape handling commands:

blocksize	Tape attachment size, in bytes
file.name	Source (T-DUMP) or destination (T-LOAD) file
item.list	List of item identifiers
mod.list	List of modifiers for special functions
records	Number of records
sel.list	List of selection criteria
seq.list	List of parameters to specify sort sequence

3.0 TCL Tape Handling

3.1 Scope: This chapter lists the tape handling verbs which are available within TCL, illustrates the format of the command, and gives a brief overview of the function performed. This information is provided for guidance purposes only, and is not intended as a complete syntactical or functional description. Detailed information regarding these verbs should be obtained from the relevant vendor documentation.

3.2 TCL Tape Handling Verbs:

S-DUMP filename {item.list} {seq.list} {sel.list} {mod.list} {HEADER "text"} {(options)}

Copies selected file items to tape, in sorted sequence.

T-ATT {blocksize}

Attaches a tape drive and establishes the blocksize.

T-BCK {records}

Backspaces tape by number of records. If records is not specified, the tape is moved back to the last previous filemark, or beginning of tape, if there are no filemarks. If records is specified, the tape will stop if it encounters a filemark or the beginning of tape. See Section 7.0 for restrictions on use of this verb.

3.2 TCL Tape Handling Verbs (continued):

T-DET

Detaches a tape drive from a process.

T-DUMP filename {item.list} {sel.list} {mod.list} {HEADER "text"} {(options)}

Copies selected file items to tape.

T-EOD

Moves tape forward to end of data.

T-FWD {records}

Moves tape forward by the number of records. If records is not specified, the tape is moved forward to the next subsequent filemark, or to the end of tape if there are no filemarks. If records is specified, the tape will stop if it encounters a filemark.

T-LOAD filename {item.list} {sel.list} {mod.list} {(options)}

Loads selected items from tape into disk file.

T-RDLBL

Reads and displays label information.

T-READ {(options)}

Reads and displays tape data record(s).

T-REW

Rewinds tape to load point.

T-WEOF

Writes a filemark on tape.

T-WTLBL {text}

Writes a tape label.

4.0 T-DUMP Tapes

4.1 Scope: The TCL verbs T-DUMP, S-DUMP, and T-LOAD provide a means of transporting selected file items from one system to another. This section describes the format of tapes produced by the T-DUMP verb and its companion S-DUMP for sorted output, and identifies the conventions to be followed which will maximize the portability across the various implementations. Other than appearing in sorted sequence, the data on an S-DUMP tape is identical to the T-DUMP version. Thus, all references to T-DUMP apply to S-DUMP, except for issues of item sequence.

4.2 File Layout: A T-DUMP file consists of a label record, zero or more data items, an end-of-file code, and a terminating filemark. The filemark implies the end of file condition, indicating that no more records are associated with this logical data file.

<Label record>

<Data item(s)>

<EOF code>

<Filemark>

4.3 Multiple Files: A tape may contain one or more logical data files, each of which follows the structure defined in the preceding paragraph. That is, each logical file consists of a label record, data record(s), its EOF marker, and a terminating filemark.

4.4 Empty Files: Files which contain no data shall be written in the follow structure:

<Label Record>

<EOF Code>

<Filemark>

4.5 Leading Filemarks: Leading filemarks on the tape are not supported in the T-DUMP format. A label record is expected to be the first block on the tape.

4.6 Trailing Filemarks: Two consecutive filemarks serve as an indication that no more files are recorded on the media.

4.7 Label Records: Label records are normally created by the T-DUMP process, and read by the T-LOAD process. The size and content of the label is described in section 6, Tape Labels, and in section 7, Hardware Capabilities, within this standard.

4.8 Block Size: The TCL command "T-ATT" sets the attachment blocksize. All data records are normally written with both physical and logical blocksize equal to the attachment blocksize. Label records are normally written with both physical and logical blocksize equal to 80 bytes. Certain exceptions are made for the characteristics of various devices and are detailed in section 7, Hardware Capabilities, within this standard.

4.9 Attachment Size: Although a wide range of physical blocksizes are possible on the various devices, the following sizes should be used for maximum portability:

1/2-inch tape	4000 bytes
1/4-inch tape	8192 bytes
Floppy diskette	500 bytes

4.10 Data Format: The format of the data records on a T-DUMP tape consist of file items, placed end to end, spanned across physical blocks as necessary. Unless the output was created with the S-DUMP verb or via a select list, the items are recorded in the same hashing sequence in which they are contained in the original source file. Special codes are used to represent the logical end of file condition. If the last data block in a file is not completely full, it is padded with a "fill" character.

4.11 Data Item Format: Generally speaking, a single item on the tape consists of the item-id terminated by an attribute mark (X'FE'), the datafield attributes of the item (including any attribute marks, value marks, and subvalue marks), followed by a buffer mark (X'FB') which ends the individual item. Graphically, this can be illustrated:

<item.id>	Item id Format: variable length character string
X'FE'	Attribute mark, terminates item id
<Attribute(s)>	Data fields within item Format: Each attribute is a variable length character string terminated by its own attribute mark, with any value or subvalue marks left in their original position
X'FB'	Buffer mark, terminates item The X'FB' immediately trails the terminating attribute mark of the last attribute in the item. Sequences of X'FEFB' cannot be embedded within items.

4.12 Items Larger than 120 Bytes: The format described above is modified slightly for items whose overall length (including item-id and all system delimiters) is greater than 120 bytes. In this case, a special two-byte buffer control code is inserted into the data after every 120 bytes of data. The buffer control code consists of a segment mark (X'FF') and a buffer mark (X'FB'). The interrupted data resumes immediately behind the two control bytes.

4.13 Pointer-Item Format: Transmitting pointer-type items via magnetic media may cause unpredictable results across SMA implementations, and should be avoided. A reliable standard for this area is yet to be determined.

4.14 Multiple Volumes: Because of the high diversity with which tape drives and disk drives detect and handle the end-of-volume condition, no single logical file should span across more than one volume when transportability is needed. If the amount of data requires more than one volume, it is advisable to subdivide the data into groups such that each group can be contained on a separate volume as an integral file.

4.15 Block Padding: As described in the paragraph on blocksizes, all physical records are written on the tape at a fixed length. Any unused buffer space behind the EOF code up to the attachment size will be filled with a buffer mark (X'FB').

On some systems, the double buffering routines in the tape drivers will cause an additional block to be written following the one containing the EOF code. This block is padded completely with the buffer mark character (X'FB').

4.16 Codes: Special codes are utilized in the T-DUMP tape format. Summarized below, they are:

_L	Identifies label record Format: segment mark (X'FF') and the character 'L'
_X	Identifies logical end of file (EOF Code) Format: segment mark (X'FF') and the character 'X'
X'FFFB'	Buffer control (after every 120 bytes) Format: segment mark (X'FF') and buffer mark (X'FB')
X'FEFB'	Item terminator. Format: attribute mark (X'FE') and buffer mark (X'FB'). Following buffer marks (X'FB') fill to attachment size.

Warning: These codes cannot be embedded within data on T-DUMP format tapes. It is especially important to note that the X'FEFB' sequence implies that X'FB' codes should not be stored as data where they can occur as the first character of an attribute.

5.0 SMA/BASIC Tapes

5.1 Scope: The SMA/BASIC programming language provides for input and output on a sequential magnetic media. This standard describes the conventions which, if followed, will maximize the portability of data between different vendor implementations.

5.2 File Layout: An SMA/BASIC file consists of zero, one, or more data records followed by a filemark. The filemark implies the end-of-file condition, indicating that no more records are associated with this logical data file.

<Data record(s)>

<Filemark>

5.3 Multiple Files: A tape may contain one or more logical data files, stacked one behind another. There is no inherent coding within the files to identify them from each other; the programs which read the data files must process them in the same order in which they were created. Each individual file must follow the structure defined in the preceding paragraph. That is, each logical file consists of data records (zero, one, or more) terminated by a filemark.

5.4 Leading Filemarks: A leading filemark at the immediate beginning of the tape will imply that the first file on the tape contains no data records.

5.5 Label Records: A standard for the writing of labels under SMA/BASIC is yet to be determined. However, SMA/BASIC will automatically bypass any existing SMA label records when a tape is read.

5.6 Block Size: The TCL command "T-ATT" sets the attachment blocksize. All data records are normally written with both physical and logical blocksize equal to the attachment blocksize. Label records are normally written with both physical and logical blocksize equal to 80 bytes. Certain exceptions are made for the characteristics of various devices and are detailed in section 7, Hardware Capabilities, within this standard.

5.7 Attachment Size: Although a wide range of physical blocksizes are possible on the various devices, the following sizes should be used for maximum portability:

1/2-inch tape	4000 bytes
1/4-inch tape	8192 bytes
Floppy diskette	500 bytes

5.8 Data Format: The expression referenced in the SMA/BASIC "WRITET" statement is written on tape in an individual block padded on the right with spaces up to the attachment blocksize. A null or oversized block invokes the "ELSE" clause, and is not written to the tape. The content of the data itself is determined exclusively by the logic of the SMA/BASIC program which generates the tape.

5.9 Multiple Volumes: Because of the high diversity with which tape drives and disk drives detect and handle the end-of-volume condition, no single logical file should span across more than one volume when transportability is needed. If the amount of data requires more than one volume, it is advisable to subdivide the data into groups such that each group can be contained on a separate volume as an integral file.

5.10 Codes: The data in an SMA/BASIC tape file is scanned for only one special character or code, the segment mark (X'FF'). The label record is identified by a segment mark (X'FF') followed by the character 'L'. The content of the data itself is determined exclusively by the logic of the SMA/BASIC program which generated the tape. However, any data following an imbedded segment mark (X'FF') in the data block is truncated during a read operation.

5.11 SMA/BASIC Tape Commands: There are four statements available in the SMA/BASIC programming language for the manipulation of tape files. Refer to the SMA/BASIC standard for syntactical and usage rules.

WRITET	Write tape
READT	Read tape
WEOF	Write end of file (filemark)
REWIND	Rewind tape

6.0 Tape Labels

6.1 Scope: This section describes the content and format of SMA tape labels.

6.2 File Layout: When present, tape labels precede the first data block in a file. No filemark automatically precedes or follows the label. Thus, the structure of a labeled file is:

<Label record>

<Data block(s)>

<Filemark>

6.3 Label Record Format: Label records are always considered to have a logical blocksize of 80 bytes. The physical blocksize and layout of the label with that physical block are detailed in section 7, Hardware Capabilities. The contents and format of the label block is indicated below. A single blank separates each field, except that two blanks separate the date and time fields.

Element	Contents
_L	Label record code Format: segment mark X'FF' and character 'L' Positions: 1-2
bbbb	Block size Format: 4 hexadecimal characters Positions: 4-7
<time>	System time when created Format: HH:MM:SS Positions: 9-16
<date>	System date when created Format: dd mon yyyy Positions: 19-29
<labeltext>	Label text Format: content depends on usage (see below) Positions: 31-76
^rr	Reel number (beginning with 01) Format: attribute mark X'FE' followed by 2 digits Positions: 78-80

6.3 Label Record Format (continued):

The format and content of the <labeltext> element within the label record depends on whether the label was created by the "T-DUMP" or "T-WTLBL" command. Label records written by the spooler use a <labeltext> element with a special format and content.

T-DUMP: When the label is created by the T-DUMP command, the <labeltext> field in bytes 31 through 76 is formatted as follows:

<filename> Name of source file from T-DUMP
Format: variable length character string, ending with a blank
Positions: 31-variable

<heading> Quoted information in HEADER option of T-DUMP
Format: variable length character string
Positions: ends in byte 76

T-WTLBL: When the label is created by the T-WTLBL command, the <labeltext> field in bytes 31 through 76 is formatted as follows:

<text> Optional text following T-WTLBL command
Format: variable length character string
Positions: 31-variable

7.0 Hardware Capabilities

7.1 Scope: This section identifies the media capabilities of the various magnetic devices supported under SMA implementations. Only those gross categories allowing off-line interchanges are considered: tape, cartridge tape, and floppy diskette. Details pertaining to the specific device usage on a given implementation should be obtained directly from the manufacturer.

This chapter also identifies the hardware characteristics and usage conventions which will maximize portability across different systems.

7.2 Hardware Standard - 1/2 Inch Magnetic Tape:

Data width (tracks per byte):
Nine

Recording density (bytes per inch):
1600 (800, 3200, and 6250 are also supported on some equipment, but not universally)

Data Block Size:
4000 bytes, logical and physical

Label Block Size:
80 bytes, logical and physical

Indicator for beginning of tape:
Reflective marker on front edge of non-recording surface of tape

Indicator for end of tape:
Reflective marker on back edge of non-recording surface of tape

Filemark indicator:
ANSI standard tape mark written and detected by the hardware

Applicable ANSI standards:
X3.39-1973

7.3 Hardware Standard - 1/4 Inch Cartridge Tape

Hardware Device Standard:
QIC 24 standard for 9 track tape controllers

Recording directions:
Serpentine, as specified in the QIC 24 standard for 9 track tapes.

Data Block Size:
Physical block size: 512 bytes (QIC 24)
Logical block size: 8192 bytes (16 physical blocks)
Logical data blocks do not include label blocks or filemark blocks.

Label Block Size:
80 bytes logical, 1 physical 512 byte block where the first 80 bytes of the physical block is the logical label record and the remaining bytes are unused.
Logical data blocks do not include label blocks.

Filemark indicator:
QIC 24 standard tape mark. Logical data blocks do not include filemarks.

TCL Verb Usage Restriction:
The verb "T-BCK" is not supported. The hardware standards do not support backward movement of the tape by record or file.

7.4 Hardware Standard - 5.25 Inch Floppy Diskette

Format:
IBM PC/XT compatible (512 byte sectors, 9 sectors per track, double-sided, double-density with 40 tracks per side, recorded at 48 tracks per inch)

Data Block Size:
500 bytes logical, 512 bytes physical, where the first 4 bytes are reserved for the filemark indicator, and the next 8 bytes are unused (and reserved), and the remaining 500 bytes are used for the logical data.

Label Block Size:
80 bytes logical, 512 bytes physical, where the first 4 bytes are reserved for the filemark indicator, the next 8 bytes are unused (and reserved), the next 80 bytes are the logical label record, and the remaining bytes are unused.

Filemark indicator:
Character string "EOF" followed by segment mark (X'FF') in the first four bytes of block (recognized by the software but not special to the hardware).

This is the last page.

✓ 6/21
PPG4

SMA STANDARD

SMA/Dictionary and Data Structure Specification

SMA: 301

March 1987

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.

Some material contained herein is designated as proprietary by individual member companies of SMA listed below. Any unauthorized use of such proprietary information is prohibited.

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

Foreword: This document provides a set of definitions for the SMA/Dictionary And Data structure. This structure, provided by all Spectrum Manufacturing Association members, is the foundation for representation of all information within SMA systems. The structure uses a set of definition items to define the various accounts and data files within the system. This document serves as a guide to the preparation of SMA/Dictionary And Data Definition Items that can be moved from one SMA system to another. For details on any specific system, the user should refer to the manufacturer's reference manual.

The SMA Executive Board wishes to thank the following individules and organzations for their contributions to the preparation of this document:

D. Harman,	Systems Management, Inc.
C. Saunders,	Fujitsu Microsystems of America, Inc.
K. Hoppe,	Altos Computer Systems
I. Sanders,	CIE Systems
H. Eggers,	Mcdonnell Douglas Computer Systems Co.
C. Wilson,	General Automation, Inc.

CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

1.0 Scope

1.1 Specification Objectives:

It is the objective of this document to provide the user with a definition of the SMA/Dictionary And Data Structure to enable the design of data files and the preparation of Data Definition Items that can be moved from one system to another with maximum portability.

1.2 Inclusions:

This document includes a definition of the SMA/Data Base system, File structure, Dictionaries and Data Definition Items, used by all the SMA systems to define the Data Base.

1.3 Exclusions:

Extensions to the Dictionary or Data structures may have been implemented by individual SMA manufacturers. This document excludes any such items which have not been adopted by a majority of the voting SMA member companies. Also, this document does not address "controlling-dependent" relationships, which are reserved for a later edition of the standard. This document does not include data structure information related to the WITHIN connective. Although many systems support "user exits", they are not considered to be within the scope of the SMA Standards.

2.0 Definitions

2.1 Names and Symbols:

The names for verbs and reserved words used in this document are used by all the SMA manufacturers. They can be easily changed for different languages without affecting the documented functionality.

The use of quotes (") and single quotes (') is required in the forms where shown.

The use of braces ({ }) means that the included string is optional.

The use of ellipsis (...) means that the preceding information can be repeated.

2.2 Structural Terms:

Item-Id - A character string (maximum 48 characters) which uniquely identifies an item within a file. The item-id may include any characters except system delimiters, however the use of blanks, single quotes, quotes, commas, and backslash characters may require special considerations.

Item Body - The variable length character structure which makes up the information content of the item. It is composed of any number of attributes, values, and subvalues. A segment mark is used to mark the end of the structure.

IDP - Indirect Data Pointer: Special type of item body that locates the data stored separately from the item body. Used to store non-character structures such as programs, as well as other extended structures.

SM - Segment Mark: Delimiter character used to mark the end of an item body. It is represented graphically by an underscore. A segment mark may not be included within data.

AM - Attribute Mark: Delimiter character used to mark the end of an attribute. It is represented graphically by an up-arrow. The last AM in an item is implied by the presence of a segment mark.

VM - Value Mark: Delimiter character used to mark the end of a value. It is represented graphically by a right bracket. The last VM in an attribute is implied by an attribute or segment mark.

2.2 Structural Terms (continued):

SVM - Subvalue Mark: Delimiter character used to mark the end of a subvalue. It is represented graphically by a backslash. The last SVM in a value is implied by a value, attribute, or segment mark.

BM - Buffer Mark: Delimiter character used to mark the beginning or ending of special character strings. It is represented graphically by a left bracket.

AMC - Attribute Mark Count: The positional count, from 1, that locates a specific attribute within the body of an item. An AMC value of zero is used to locate the item-id.

VMC - Value Mark Count: The positional count, from 1, that locates a specific value within a multivalued attribute.

SVMC - Subvalue Mark Count: The positional count, from 1 that locates a specific subvalue within a multisubvalued structure.

3.0 Dictionary And Data Structure

3.1 System File Structure:

The SMA systems use the SMA-based file pointer system which is a hierarchical structure consisting of up to four levels.

The top level, known as level zero, is the System Dictionary file and contains the File Definition Items for the Master Dictionary file of each of the Accounts on the system. This file also contains the File Definition Items for system wide files.

Level one, known as the Master Dictionary (MD) level file for an account, contains File Definition Items, File Synonym Items, Data Definition Items, stored procedures, verbs, and vocabulary words for the SMA/Retrieval language.

Level two, known as a Dictionary Level file, contains the File Definition Items for data portions of the file and optionally Data Definition Items.

Level three, known as the data level file, contains data items. File Definition Items and File Synonym Items, if present, will be ignored if accessed at the data level.

3.2 File Definition Item:

The File Definition Item is stored in the dictionary that is associated with the file.

The File Definition Item has the following form.

AMC	Description
Item-Id	The name by which the file is referenced.
1	Defines the item to be a pointer to a file. It must contain one of the following forms:
	D — The file is of standard form containing items that are saved on a file save operation.
	DX — The file is the same as a D file, except that the file will not be saved on a file save operation.

3.2 File Definition Item (continued):

AMC	Description
1	Forms of File Definition Item type, continued:
	DY The file is the same as a D file, except that only valid File Definition Items will be saved on a file save operation; other types of items will not be saved. Note: File Definition Items are only valid in file levels Ø, 1, and 2. When the file save operation encounters any valid File Definition Item, it proceeds to save the indicated file.
	DC The file is the same as a D file, but the file may also contain indirect data pointer items that are saved on a file save operation.
	DCX The file is the same as a DC file, except that the file will not be saved on a file save operation.
	DCY The file is the same as a DC file, except that only valid File Definition Items will be saved on a file save operation; other types of data will not be saved. Note: File Definition Items are only valid in file levels Ø, 1, and 2. When the file save operation encounters any valid File Definition Item, it proceeds to save the indicated file.
2	Base . . . . | These three attributes define the physical file structure and must not be modified by the user.
3	Modulo . . . |
4	Separation . |
5	The access lock codes used for access protection, represented in a multi-valued list.
6	The update lock codes used for update protection, represented in a multi-valued list.

3.2 File Definition Item (continued):

AMC	Description
7	External processing codes used to convert the item-id between the processing format and the external format. Multiple processing codes can be used and are separated by value marks.
8	Reserved.
9	Defines the justification of data in the output form of the element. This attribute, if included, must be an "L", "R", or "U". The code is used in formatting the output, and in determining the sort sequence when sorting the data and is: L,U Specifies a left-to-right sort, and will left-justify, without folding. This may cause the field to be overlayed by the next attribute. R Specifies a right-justified numeric sort (including alphanumeric elements) which may overlay the previous attribute. Default if not specified is L. File Definition Items for level one files may contain combinations and/or additional codes. These codes are used for other system processors and their meaning is specified elsewhere. If multiple codes are present, then only the first code is used for SMA/Retrieval purposes.
10	Defines the maximum column width when displaying the item-id. The modifier ID-SUPP may be used to suppress the output of the item-id. Default value is 9.
11	Reserved.
12	Reserved.
13	Reallocation parameter which is of the form: (new.modulo,new.separation) where the new.modulo and new.separation will be used for attributes 3 and 4 upon the file create as a part of a file restore.

Note that AMC 5-13 are optional.

3.3 File Synonym Item:

A file may be referenced by the use of a file synonym. Multiple file synonyms can exist for the same data file and may be stored in the three dictionary levels.

The File Synonym Item has the following form:

AMC	Description
Item-id	The name by which the file is referenced.
1	The 'Q' indicates that this is a File Synonym Item.
2	The name of the account in which the the file has been defined. See table below.
3	The name of File Definition Item of the file defined. See table following AMC 10.
4	Reserved.
5	Reserved.
6	Reserved.
7	External processing codes used to convert the item-id between the processing format and the external format. Multiple processing codes can be used and are separated by value marks.
8	Reserved.

3.3 File Synonym Item (continued):

AMC	Description
9	Defines the justification of data in the element. This attribute, if included, must be an "L", "R",or "U". The code is used in formatting the output, and in determining the sort sequence when sorting the data and is: L,U Specifies a left-to-right sort, and will left-justify, without folding. This may cause the field to be overlayed by the next attribute; R Specifies a right-justified numeric sort (including alphanumeric elements) which may overlay the previous attribute. Default if not specified is L. File Synonym Items for level one files may contain combinations and/or additional codes. These codes are used for other system processors and their meaning is specified elsewhere. If multiple codes are present, then only the first code is used for SMA/Retrieval purposes.
10	Defines the maximum column width when displaying the item-id. The modifier ID-SUPP may be used to suppress the output of the item-id. Default value is 9.

Note that AMC 2-10 are optional.

3.3 File Synonym Item (continued):

Reference table for attributes 2 and 3 in File Synonym Items:

AMC-2	AMC-3	Description
account	file	Form used to reference a file in a specified account. It may be this account or some other account.
null	file	Form used to reference a file in this account by another name.
null	null	Form used to reference the dictionary of this file without the use of DICT.
account	null	Form used to reference the master dictionary of another account.

If attributes 7, 9, and 10 exist in the file synonym definition, they take precedence over those attributes in the File Definition Item when referencing the file via the file Synonym Item.

3.4 Data Definition Item:

Data Definition Items are used to define the data structure of the associated data file. The item-ids of these items are used in the SMA/Retrieval Language sentences in selection criteria, sort criteria, and output criteria.

Data Definition Items have the following structure:

AMC	Description
Item-Id	The name by which this data definition is referenced.
1	An 'A', 'S', or 'X' identifies the item as a Data Definition Item. Note that 'A' and 'S' have identical functionality but may be used by the application designer to identify primary 'A' or synonym 'S' attribute data definitions. 'X' is used as place holder for a Data Definition Item in numeric default group of Data Definition Items. The value of an attribute with a Data Definition Item containing an 'X' will not be output. Attribute Data Definition Items with a 'X' should never be explicitly named in a SMA/Retrieval language sentence.
2	The numeric value (AMC) locating the attribute in the item which is being defined.
3	Textual data used as a column heading in LIST or SORT sentences. If null, the item-id is used as the heading. May contain blanks for formatting. The reserved character "\" is used to specify a null heading. Multiple line headings for columnar listings may be specified by storing multiple values.
4	Defines the 'controlling-dependent' relationship.
5	Reserved.
6	Reserved.
7	External processing codes used to convert between the processing format and the external format. Multiple processing codes can be used and are separated by value marks.

3.4 Data Definition Item (continued):

AMC	Description
8	Internal processing codes used to convert from internal format to processing format. Multiple processing codes can be used and are separated by value marks.
9	Defines the justification of data in the element. This is a required attribute, and must be an "L", "R", "T", or "U". The code is used in formatting the output, and in determining the sort sequence when sorting the data and is: L Specifies a left-to-right sort, and will left-justify, folding long strings at the end of the column width defined by attribute 10; R Specifies a right-justified numeric sort (including alphanumeric elements); T Specifies a left-to-right sort, and will left-justify, folding long strings at blanks; U Specifies a left-to-right sort, and will left-justify, without folding. Default if not specified is L.
10	Defines the maximum column width when displaying the attribute. A value of zero may be used to suppress output on detail lines. The default value is 9.

Note that AMC 2-10 are optional.

3.5 Item Structure:

An item is said to be made up attributes, each of which may be made up of values, each of which may be made up of subvalues.

An element refers to the data in an attribute, value, or subvalue. It may be null, a numeric string, or a character string.

3.5.1 Item-Id:

Each item within a file has associated with it a unique item-id. This item-id may be referenced as attribute Ø within Data Definition Items and processing codes.

3.5.2 Attribute:

An attribute is a data element within an item. Attributes are sequentially numbered starting from one and are delimited by attribute marks. A given attribute typically contains data with the same context in all the items in a particular file. The SMA/Retrieval language assumes that all items participating in a particular sentence/report contain the same context of data within any given attribute.

3.5.3 Multi-Valued Attributes:

An attribute containing one or more value marks is said to contain multi-values, sequentially numbered starting from one.

3.5.4 Multi-Valued Values:

A value containing one or more subvalue marks is said to contain multi-subvalues, with the subvalues sequentially numbered starting from one.

4.0 Processing Codes

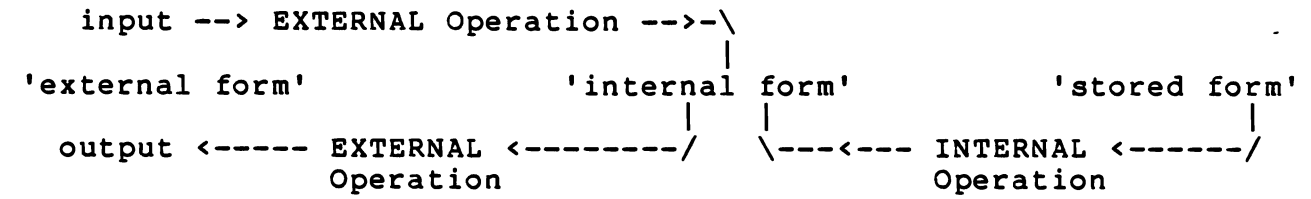
The SMA/Dictionary and Data Structure provides a set of processing codes that can be specified in attribute 7, where they perform EXTERNAL operations, or in attribute 8, where they perform INTERNAL operations.

During the processing of a SMA/Retrieval language sentence, the data in the items can exist in three different forms. The first is the 'stored' form. Whenever the element is retrieved from an item it is picked up in the stored form.

The INTERNAL operations, if specified, are applied to the element, converting it from the 'stored' form to the 'internal' form. The 'internal' form of the Data Definition Item is used whenever it:

1. is part of a sort.criteria,
2. is compared to a selection.criteria,
3. is compared for an output limiter,
4. is used for a TOTAL or GRAND-TOTAL computation,
5. produces a control break,
6. is printed, except on break lines,
7. is output by reformat or select output.criteria.

The EXTERNAL operations, if specified, are applied to convert an element between 'internal' form and 'external' form. The 'external' form is used for user input and output. If a Data Definition Item contains an EXTERNAL operation and it is followed by selection.criteria in a SMA/Retrieval Language sentence, then the EXTERNAL operation is applied as an input conversion. The select.criteria value is converted from 'external' form to 'internal' form. Print.limiters and explosion.limiters are converted in a similiar manner. This action causes selection, sorting, and limiting to always operate on the 'internal' form of the data elements.



4.1 Processing Codes Summary:

CODE	DESCRIPTION
A	ALGEBRIAC. Used to evaluate algebriac expressions.
C	CONCATENATE. Used to concatenate elements.
D	DATE. Used to convert dates.
F	FUNCTION. Used to manipulate elements.
G	GROUP. Used to extract one or more fields separated by a given non-system delimiter.
L	LENGTH. Used to validate the length of an element.
MC	MASK CHARACTER. Used to convert strings to upper or lower case, or to extract alphabetic or numeric characters from strings.
ML	MASK DECIMAL. Used to format and scale numbers, left justified.
MR	MASK DECIMAL. Used to format and scale numbers, right justified.
MT	MASK TIME. Used to convert time.
MX	MASK HEXADECIMAL EXPANSION. Used to convert ASCII characters to their hexadecimal representations.
MY	MASK HEXADECIMAL COMPRESSION. Used to convert hexadecimal characters to their ASCII representation.
P	PATTERN MATCH. Used to validate elements against a specified pattern.
R	RANGE. Used to validate elements which fall within the specified numeric ranges.
S	SUBSTITUTION. Used to generate alternative data for the referenced element.
T	TEXT EXTRACTION. Used to extract a fixed field from an element.
Tfile	FILE TRANSLATION. Uses the element as an item-id to access attributes in the specified file.

4.2 Algebriac Processor, A CODE:

The 'A' code provides functions similar to the 'F' code, and is written in an algebriac form using infix notation. For example, A1+2 would add the element from attribute 1 to the element of attribute 2. Evaluation of the expression proceeds from left to right unless reordered by use of parentheses. The infix operators have no order of precedence defined. The form is:

A expression {operator expression}...

where:

expression = operand {operator operand}

Parenthesis may be used as required to control the order of expression evaluation. The inner-most parenthesis expression will be evaluated first. The term "expression-1" is used to refer to the operand or expression on the left side of an operator, and the term "expression-2" to the operand or expression on the right side.

The permissible operands are:

amc{R{R}} An Attribute Mark Count specifies the number of the attribute from which the element is to be retrieved for use in the operation. If the AMC is followed by R, it specifies that the first value of an attribute is to be used repeatedly when evaluating with other multi-valued attributes. If a second R is present, the first subvalue is used repeatedly for evaluation with other multi-subvalued attributes.

N(name) The character N followed by the name of a Data Definition Item enclosed in parentheses can be used to specify the element to be used in the operation. The 'name' must exist in the dictionary being used. The data element retrieved will be specified via attribute 2 unless the Data Definition Item referenced contains a function. If this occurs, the function will be performed.

literal A alpha-numeric literal string is specified by being enclosed in either single quotes or double quotes.

4.2 Algebriac Processor, A Code (continued):

The following special system counters and values may be used as operands in an A-code expression. Refer to the section on F-code for further details. They are:

NI	the item counter
NV	the value counter
NS	the subvalue counter
ND	the detail line counter
NB	the break level counter
D	the system date (in internal format)
T	the system time (in internal format)

There are several special functions which may be used as operands in the A-code expression. They are:

R(exp,exp)	The Remainder function takes two expressions as operands, and returns the remainder of the first operand divided by the second.
S(exp)	The Summation function computes the summed total of the enclosed expression for all elements of a multivalued or multisubvalued set.
expl[exp2,exp3]	A sub-string of an element is specified. Expl is the element from which the substring is to be extracted, exp2 is the character in the element from which to start the extraction, and exp3 is the number of characters to be extracted.

4.2 Algebriac Processor, A Code (continued):

The arithmetic and relational operators all require two operands. The arithmetic operators are:

+	adds the two expressions and returns the sum.
-	subtracts expression-2 from expression-1 and returns the difference.
*	multiplies the two expressions and returns the product.
/	divides expression-1 by expression-2 and returns the quotient. The quotient is always an integer but may have implied scaling or a decimal point.
:	concatenates the second operand onto the end of the first operand. The operands are considered as character strings and the result is a character string.

The relational operators return a one if the relational operation evaluates as true and a zero if the relational operation evaluates a false. The relational operators are:

=	Expression-1 equal to expression-2.
#	Expression-1 not equal to expression-2.
<	Expression-1 less than expression-2.
>	Expression-1 greater than expression-2.
>=	Expression-1 greater than or equal to expression-2
<=	Expression-1 less than or equal to expression-2.

In the absence of parentheses to indicate the order in which operators are to be applied, operations proceed in straight left to right sequence, with no precedence among operators.

4.3 Concatenation, C Code:

The 'C' code provides the facility to concatenate attributes and/or literals prior to output. The form is:

C;op{x op}...{x}

x — is the character to be inserted between the concatenated attributes and/or literals. A semicolon is a reserved character that means no separation character is to be used. Any non-numeric (except system delimiters) is valid, including a space.

op — is the attribute mark count (AMC); or any string enclosed in single quotes, double quotes, or backslashes; or an asterisk, which specifies the last generated value from a previous operation is to be used.

4.4 Date Conversion, D Code:

The 'D' code provides for the conversion of dates in internal format to external for output or from external format to internal format when used with selection.criteria. December 31, 1967 is defined as day zero with positive values following and negative values earlier. Many of the date conversions will not operate on input data as they do not uniquely define a month, day, and year. The form is:

D{n}{xm}{s}

n — is an option single digit number which specifies the number of digits to occur in the year on output. If 'n' is 0, no year will appear in the the date. The only valid digits for 'n' are 0,1,2,3, or 4. If 'n' is not specified then n = 4 is assumed.

x — Stands for any single non-numeric character which specifies delimiter between fields for group extract. The 'x' cannot be one of the system delimiter.

m — is a single digit number that must accompany 'x' (if 'x' is specified). 'm' specifies the number of fields to skip for group extraction. The group extraction is done before the conversion is performed.

4.4 Date Conversion, D Code (continued):

s is either any non-numeric character that may be specified to separate the day, month, and year on output, or special date sub-code. If 's' is specified, the output format will be MMsDDs{{{{Y}Y}Y}Y}. If 's' is not specified, the date output format will be DD MMM {{{{Y}Y}Y}Y}. (MM is a two digit month, MMM is a three character alpha month abbreviation.) On External to Internal conversion of a two digit year, the years range from 1930 through 2029.

The permissible date sub-codes are:

D	Day of the month.
I	Internal format, Reverse conversion.
J	Julian day of year.
M	Month numeric.
MA	Month alphabetic.
Q	Quarter numeric.
W	Weekday numeric (Monday=1, Sunday=7).
WA	Weekday alphabetic.
Y	Year. Default = 4 digits.

4.5 Function Processor, F Code:

The 'F' code processor uses a 15 element post-fix push-down stack for storing values. An operation specified by an F-code operates on the last one, two or three entries pushed onto the stack. Entries are removed from the stack as they are used in the operation. The results of the operation is pushed onto the stack. This continues for each operator until the entire F-code is processed. The final result is then the value on the top of the stack. The form is:

FS;elm{;elm...}

Note that this form of the 'F' code processor differs from previous implementations in the use of the S to designate standard form. This standard form includes all ordered binary operations in classic reverse Polish ordering, including the comparison operations.

4.5 Function Processor, F Code (continued):

where 'elm' may be any of the following:

amc{R{R}}	A numeric Attribute Mark Count specifying the element to be pushed onto the stack. If the AMC is followed by R, it specifies that the first value of an attribute is to be used repeatedly when evaluating with a multi-valued attribute. If the second R is present, it specifies that the first subvalue of a value is to be used repeatedly.
Cn	A capital 'C' followed by a string, specifies that the string is to be pushed onto the stack. The string is ended by the next semicolon.
D	specifies that the current date is to be pushed onto the stack (internal format).
literal	The literal string enclosed in either single or double quotes is pushed onto the top stack entry.
T	specifies that the current time is to be pushed onto the stack (internal format).
NA	specifies that the number of attributes in the item is to be pushed onto the stack.
NB	specifies that the current Break level number is to be pushed on to the stack. 1 = the lowest level break and 255 = the grand-total line.
ND	specifies that the number of items since the BREAK on a break line is to be pushed onto the stack. If on a GRAND-TOTAL line, it equals the item count.
NI	specifies the value of the current item counter is to be pushed onto the stack (number of items listed or selected).
NL	specifies that the length of the item is to be pushed onto the stack.
NS	specifies the current subvalue counter, for columnar listing only, is to be pushed onto the stack.
NV	specifies the current multi-value counter, for columnar listing only, is to be pushed onto the stack.

4.5 Function Processor, F Code (continued):

where 'elm' may be any of the following:

LPV	specifies the loading of the value from the previous processing code.
*{n}	Multiplication of the top two stack entries. If 'n' is specified, the result is divided by 10 raised to the power of n.
/	Divide the second stack entry by the top stack entry and replace the top stack entry with the quotient.
R	Divide the second stack entry by the top stack entry and replace the top stack entry with the remainder
+	Add the second stack entry to the top stack entry and replace the top stack entry with the sum.
-	Subtract the top stack entry from the second stack entry and replace the top stack entry with the difference.
:	The top stack entry is concatenated onto the end of the second stack entry, and the resulting concatenated string replaces the the top stack entry.
[]	A subset from the third stack entry is extracted, using the second stack entry as the starting character position, and the top stack entry as the number of characters to be extracted; the result is placed in the top stack entry.
S	A total sum of all previous computation is placed on the top of the stack. The sum operator is used with multi-valued or multi-subvalued elements to produce a single value. Multiple S operators may be present within a function. The domain of a function begins at either the start of the function or immediately following the previous S operator. At the conclusion of the S operator, a single value is present on the stack.
_	Exchange the top two stack entries.
P	Duplicates the top stack entry back on to the stack.

4.5 Function Processor, F Code (continued):

where 'elm' may be the following:

(...) A standard conversion operator, enclosed in parentheses, will operate on the top stack entry and the result will replace the original top stack entry.

The following relational 'elm's operate on the top two stack entries, and a result of zero or one is placed in the top stack entry, depending on whether the condition is not or is satisfied.

= Stacks a one if the two top stack entries are equal, and a zero is stacked if they are unequal.

Stacks a one if the two top stack entries are unequal, and stacks a zero if they are equal.

> Stacks a one if the second stack entry is greater than the top stack entry, stack zero otherwise.

< Stacks a one if the second stack entry is less than the top stack entry, a zero otherwise.

] Stack a one if the second stack entry is greater than or equal to the top stack entry, a zero otherwise.

[Stacks a one if the second stack entry is less than or equal to the top stack entry.

4.6 Group Extraction, G Code:

The 'G' code provides the facility to extract from an element one or more contiguous fields separated by a given delimiter. The form is:

G{m}xn

m specifies the number of fields to skip. If m is not specified, zero is assumed, and no fields are skipped.

x represents any single non-numeric character, except any system delimiter, which is the field separator.

n is a decimal number which is the number of contiguous fields to be extracted.

4.7 Length, L Code:
The 'L' code validates the length of an element. The form is:

L{n{,m}}

L	returns the length of the element.
Ln	returns the element if it is less than or equal to 'n' characters long, otherwise a null is returned.
Ln,m	returns the element if it is equal to or greater than 'n' characters long and less than or equal to 'm' characters, otherwise a null is returned.

4.8 Mask Character, MC Code:
The 'MC' code provides the facility to change an element to upper or lower case, to select out certain classes of characters, or convert from hexadecimal to decimal and from decimal to hexadecimal. The forms are:

MCA	Extracts all alphabetic characters from an element.
MC/A	Extracts all non-alphabetic characters from the element.
MCD{X}	Converts decimal element to a hexadecimal element.
MCL	Converts an element to lower case. Will convert all upper-case letters to lower-case; has no effect on lower-case letters or non-alphabetic characters.
MCN	Extracts all numeric characters (0-9) from an element.
MC/N	Extracts all non-numeric characters form an element.
MCT	Converts first letter following a non-alphabetic character to upper-case. The characters up to the next non-alphabetic character are converted to lower-case. This process is repeated through-out the entire element.
MCU	Converts an element to upper case. Will convert all lower-case letters to upper-case; has no effect on upper-case letters or non-alphabetic characters.
MCX{D}	Converts hexadecimal element to a decimal element.

4.9 Mask Decimal, ML And MR Codes:

The 'ML' and 'MR' codes provide the facility to do special output formatting of data elements. The ML code specifies that the output is to be left justified and the MR code specifies that the output is to be right justified. The forms are:

```
ML{n{m}}{Z}{,}{cr}{$}{(format-string)}
MR{n{m}}{Z}{,}{cr}{$}{(format-string)}
```

n is a single decimal digit (Ø-9) which specifies the number of digits to be printed to the right of the decimal point. If 'n' is not specified, Ø assumed. If Ø is assumed or specified, no decimal point is printed.

m is a single digit numeric (Ø-9) which specifies that the element is to be divided by that power of ten. The number 'm' is the number of implied digits to the right of the decimal point. If m > n, then the element will be rounded to 'n' digits.

Z specifies zero-suppression. An element of Ø (zero) will be printed as blanks.

, specifies the insertion of a comma every three digits to the left of the decimal point.

cr specifies the designation of debit/credit symbols as:

- C causes negative elements to be followed by the letters CR.
- D causes positive elements to be followed by the letters DB.
- E causes negative elements to be enclosed inside angle brackets.
- M causes negative elements to be followed by a minus sign.
- N causes the minus sign on negative elements to be suppressed.

In the absence of a credit symbol, negative numbers are presented with a leading minus sign.

$ causes a currency symbol to be appended to the front of the element before justification.

4.9 Mask Decimal, ML And MR Codes (continued):

The format mask specification, which is enclosed in parentheses, consists of format codes and literal data. The format codes are one of the characters #, *, or %, optionally followed by a number to specify that number of repetitions of the characters. The meaning of the format codes are:

#{n}	specifies that the element is to be justified in a field of 'n' blanks.
*{n}	specifies that the element is to be justified in a field of 'n' asterisks.
%{n}	specifies that the element is to be justified in a field of 'n' zeroes.

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 %).

4.10 Mask Time Conversion, MT Code:

The 'MT' code provides the facility for converting times to or from internal format. The internal time format is the number of seconds. The form is:

MT{H}{S}

H	is the capital letter H, which specifies 12 hour format. If 'H' is omitted, 24 hour (international) format is assumed.
S	is the capital letter S, which specifies seconds on on output. If 'S' is omitted seconds are not listed on output.

When the codes MTH or MTHS are used, AM or PM is always displayed immediately following the 12 hour time.

4.11 Mask Hexadecimal Expansion, MX Code:

The 'MX' code provides the facility to convert an element, one byte at a time, into the corresponding hexadecimal representation. Each character will be converted to a 2-byte hexadecimal number. On input conversion, the element will be considered right justified. The form is:

MX

4.12 Mask Hexadecimal Compression, MY Code:

The 'MY' code provides the facility to convert a hex element, two bytes at a time, into the corresponding ASCII representation. Each character will be converted to a 1-byte ASCII character. The form is:

MY

4.13 Pattern Matching, P Code:

The 'P' code validates an element if it matches any of the specified patterns. If the element does not match any pattern, a null is returned. The form is:

P(op){;(op)}...

Any combination of the following forms is valid within an 'op'.

op	Description
nN	The integer number 'n' followed by the letter 'N', tests for n numeric characters.
nA	The integer number 'n' followed by the letter 'A', tests for n alpha characters.
nX	The integer number 'n' followed by the letter 'X', tests for n characters.
'literal'	A literal, enclosed in single quotes ('), tests for the literal string.

4.14 Range, R Code:

The 'R' code validates an element which falls within a specified range. The form is:

Rn,m{;n,m}...

n is the starting integer of the range.

m is the ending integer of the range.

If the range specifications are not met, null is returned.

4.15 Substitution, S Code:

The 'S' code substitues the element of the reference attribute with the element of the first specified attribute or a literal if the element of the reference attribute is not null or zero. If the element of the reference attribute is null or zero, then it will be substitued with the element of the second specified attribute or a literal. The form is:

S;op1;op2

op1 if the element is not null or zero, then this attribute or literal (enclosed in single quotes) is used for substitution.

op2 if the element is null or zero, then this attribute or literal (enclosed in single quotes) is used for substitution.

Note that an asterisk, which specifies the last generated value from a previous operation, may be used as either 'op1' or 'op2'.

4.16 Text Extraction, T Code:

The 'T' code extracts a contiguous string of characters from an element. The form is:

T{m,}n

m is the optional starting column number.

n is the number of characters to be extracted.

The form 'Tm,n' counts columns and extracts characters from left to right of the element, regardless of the code in attribute 9 of the Data Definition Item.

The form 'Tn' extracts 'n' characters either from the left or the right, depending upon on the code in attribute 9 of the Data Definition Item. If that code is not 'R', then the 'Tn' form extracts the first 'n' characters of the element. If that code is 'R', then the 'Tn' form extracts the 'n' rightmost characters of the element.

4.17 File Translation, Tfile Code:

The 'Tfile' code provides the facility for converting an element by translating through a file. The element to be translated is used as an item-id for retrieving an item from the specified translation file. The translated element is retrieved from the specified attribute of the item. The form is:

T{DICT }file;c{n};{i-amc};{o-amc{;b-amc}}

DICT specifies the use of the dictionary of the file instead of the data portion of the file.

file specifies the name of the file to be used for the translation.

c is the translation sub-code, which is one of the following:

- V Conversion item must exist on file, and the specified attribute must have an element. Aborts with an error message if translation is impossible.
- C Convert if possible; use original element if item in translate file does not exist or has a null conversion attribute.
- I Input verify only - functions like 'V' for input and like 'C' for output.
- O Output verify only - functions like 'C' for input and like 'V' for output.
- X Convert if possible, otherwise return a null element.

n is a value mark count. If 'n' is specified, then only the element in value n will be returned. If 'n' is not specified, then all the values in the element are concatenated together with blank separators and returned.

i-amc is the decimal attribute number for input translation. If the i-amc is omitted, no input translation takes place.

o-amc is the numeric attribute number for output translation.

b-amc if specified, will be used instead of o-amc during the listing of break-on and total lines.

THIS IS THE LAST PAGE

✓ PP64
6/21/88

SMA STANDARD

SMA/RETRIEVAL Language Specification

SMA: 401

January 1988

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.

Some material contained herein is designated as proprietary by individual member companies of SMA listed below. Any unauthorized use of such proprietary information is prohibited.

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/RETRIEVAL language. This language, provided by all Spectrum Manufacturers Association members, is a report generating process which enables quick and easy preparation of various listings and queries from the Data Base. The language uses a limited natural language sentence format for accessing the Data Base. This document is intended to serve as a guide to the preparation of SMA/RETRIEVAL language sentences. For details on any specific system, the user should refer to the manufacturer's reference manual.

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

D. Harman, Systems Management, Inc.
C. Saunders, Fujitsu Microsystems of America, Inc.
K. Hoppe, Altos Computer Systems
I. Sandler, CIE Systems
H. Eggers, Mcdonnell Douglas Computer Systems Co.
C. Wilson, General Automation, Inc.

CONTENTS

CONTENTS

1.0 Scope

1.1 Specification Objectives:

It is the objective of this document to provide the user with a defined syntax of the SMA/Retrieval Language to enable preparation of SMA/Retrieval statements that can be moved from one system to another with maximum portability.

1.2 Inclusions:

This document includes all the commonly available verbs with syntactical representation to clearly define the results produced by each.

1.3 Exclusions:

Extensions to the SMA/Retrieval language have been implemented or proposed by several SMA members. This document excludes any such items which have not been adopted by the majority of the SMA member companies. For further details on any differences between manufacturer's systems, see the manufacturers documentation.

2.0 Definitions

2.1 Names and Symbols:

The names for verbs and predefined words used in this document are used by all the SMA manufacturers. They can be easily changed for different languages without affecting the documented functionality.

The use of quotes (") and single quotes (') is required in the forms shown below.

The use of braces ({ }) means the included string is optional.

The use of ellipsis (...) means the preceding information can be repeated.

2.2 Structural Terms:

Item-Id - A character string (maximum 48 characters) which uniquely identifies an item within a file. The item-id may include any characters except system delimiters. The use of blanks, single quotes, quotes, commas, and backslash characters may require special considerations.

Item Body - The variable length character structure which makes up the information content of the item. It is composed of any number of attributes, values, and subvalues.

SM - Segment Mark: Delimiter character used to mark the end of a data structure.

AM - Attribute Mark: Delimiter character used to mark the end of an attribute. The last AM in an item is implied by the presence of a segment mark.

VM - Value Mark: Delimiter character used to mark the end of a value. The last VM in an attribute is implied by an attribute or segment mark.

2.2 Structural Terms (continued):

SVM - Sub-value Mark: Delimiter character used to mark the end of a subvalue. The last SVM in a value is implied by a value, attribute, or segment mark.

AMC - Attribute Mark Count: The positional count, from 1, that locates a specific attribute within the body of an item. An AMC value of zero is used to locate the item-id.

VMC - Value Mark Count: The positional count, from 1, that locates a specific value within a multivalued attribute.

SVMC - Sub-value Mark Count: The positional count, from 1 that locates a specific subvalue within a multisubvalued structure.

BM - Buffer Mark: Delimiter character used to mark the beginning or ending of special character strings.

3.0 SMA/Retrieval Language Sentence

3.1 SMA/Retrieval Language Sentence General Form:

The SMA/Retrieval Language invokes processing of data from a file according to specified criteria by the use of a single sentence.

The general form of the sentence takes up one logical line.

```
verb    {file.modifiers} file.name
        {item.list/item.id.selection.criteria}
        {USING {DICT} file.name}
        {data.definition.item.selection.criteria}
        {sort.criteria}
        {output.criteria}
        {modifiers} {(option,...)}
```

Only the verb and file name are mandatory. Blanks are used as separators between the parts of the sentence.

The SMA/Retrieval Language comes with a standard set of verbs, modifiers, and relational operaters. These words are defined as items in the user's Master Dictionary (MD). The user may define any number of synonyms for these words, and remove the supplied entries, thereby creating his own semantics for the language. Thus the SMA/Retrieval language can be tailored to fit any language.

3.2 Verb:

A verb specifies what processing will be performed on the file. The verb must be the first word in the sentence.

3.3 File.Modifier:

Certain modifiers are valid as modifiers of the file name, and if present, must precede the file name.

3.3.1 DICT:

The file modifier DICT stipulates that the dictionary section of the named file contains the data to be operated on.

3.3.2 ONLY:

The file modifier ONLY stipulates that only the item-ids of the items shall be output, suppressing any existing "default" output.criteria.

3.4 File.Name:

The file name stipulates the file to be operated on by the verb. It also implies the dictionary to be used as the source of data.definition.items, unless the USING clause appears later in the statement.

3.5 USING:

The USING connective stipulates that the following named file is to be used as the source of data.definition.items during processing instead of the dictionary of the file to be processed. If the modifier "DICT" precedes the file.name then the dictionary of the file is to be used, otherwise the data portion of the file is to be used as the source.

3.6 Item.Lists

Either an Explicit or an Implicit item list is used for retrieving items from a data file for processing. If no explicit list of item-ids is specified, and an implicit.list is not active, all items in the file will be used by the verb. Only those items within the list, that are present in the file, will be used by the verb.

3.6.1 Explicit.Item.List:

The explicit.item.list is a list of item-ids, each of which is surrounded by single quotes ('), to be operated on by the particular verb.

If present, an explicit list takes precedence over an implicit list, and causes the implicit list to be ignored.

3.6.2 Implicit.Item.List

An implicit.item.list is a list which has been activated by a previous command such as SELECT or GET-LIST.

3.7 Selection.Criteria:

Items may be stipulated as members of the set output by the inclusion of the selection.criteria in the command.

A selection criterion may be either an item.id.selection.criteria or a data.definition.item.selection.criteria.

A selection criterion is made up of a relational operator and a value.string.

3.8 Item-id Selection.Criteria:

Selection.criteria are placed on the item-id by following the file.name with one or more relational.connective and character.string pairs, ANDed together as desired. There must be at least one relational.connective in the sequence of pairs, otherwise the collection of 'Item-id' and "character.string" elements will be treated as a list of explicit Item-ids.

Any 'Item-id' or "Character.string" elements not preceeded by a relational.connective will be taken to be "equal"tests.

The forms 'Item-id' and Character.string" have the same status in the list immediately following the file.name and preceeding the first data.definition.item and are called Item.id.values.

The Item-id selection.criteria form is:

```
relational.connective item.id.value
   {{AND/OR} {relational.connective} item.id.value}...
where:
      item.id.value is

           'Item-id'
      or
           "Character.string"
```

An item must pass the item-id selection.criteria before any other criteria are considered. Therefore, item-id selection.criteria are implicitly ANDed with all other selection.criteria, and an explicit AND is illegal after item-id selection.criteria.

3.8.1 ORing Item.Id.Values:

Several item.id.values in succession are taken to be ORed with implicit equal relational.connectives; that is, if an item-id matches any of the item.id.values, the item passes this criterion.

3.8.2 ANDing Item.Id.Values:

The AND evaluation connective may be placed between item.id.values. In this case, an item-id must pass all the tests stipulated by the relational.connective-item.id.value pairs for the item to pass this criterion. Note that use of the implicit equal relational.connective will define a criterion that can not be passed by any item-id, as an item-id can not be "equal" to two different values.

3.9 Data.Definition.Item Selection.Criteria:
Items may be selected based on the generated contents of stipulated data.definition.items.

Stipulation of data.definition.item selection criteria requires the use of the WITH connective.

The form of the data.definition.item selection criterion is

```
WITH {EACH} {NO} data.definition.item.name
   {value.string.criteria} {{AND/OR} WITH {EACH} {NO}
   data.definition.item.name {value.string.criteria}}...
```

The WITH connective may be associated with the EACH and/or with the NO modifier. The effect of the inclusion of the EACH modifier is that each multi-value of the attribute stipulated by the data definition item must pass the value.string criterion. The effect of the NO modifier is to pass data.definition.items whose value is null if there is no value.string.criterion, or to pass those data.definition.items which do not pass the value.string criteria if one is present.

A command may contain at least 9 ORed data group selections.

3.9.1 ORing Data.Definition.Items:
A sequence of data.definition.item selection criteria will be implicitly ORed together if they are not explicitly ANDed together. If an item passes any one of ORed data.definition.item selection criteria it will be accepted.

3.9.2 ANDing Data.Definition.Items:
Inserting an AND between the end of a data.definition.item selection criterion and the beginning of another has the effect of ANDing the criteria together. Any number of data.definition.item selection criteria may be ANDed together to form a group. An item must pass all of the ANDed criteria in order to be accepted.

There may be as many as 9 of these groups of more than one data.definition.item selection criteria which have been ANDed together. These groups are implicitly ORed together.

3.10 Value.String.Criteria:
The form of a value.string.criterion is:

```
{relational.connective} "Character.string" {{AND/OR}
   {relational.connective} "Character.string"} ...
```

If there is no relational.connective preceeding a "Character.string", then the test defaults to "equal". A data value will pass an ORed group of value.string criteria if it passes any one. A data value will pass an ANDed group of value.string criteria if it passes all entries in the group. The logic of evaluations with NOT follows as with the NOT evaluation for Item-id Selection Criteria, that is, the AND connective must be used to exclude more than one value.

3.10.1 Relational Connectives:

The allowable relational.connectives are:

mnemonic	symbol	meaning
EQ	=	equal
NE	#	not equal
LT	<	less than
LE	<=or=<	less than or equal
GT	>	greater than
GE	>=or=>	greater than or equal

The lack of a relational.connective defaults to the EQ (=) relational.connective. Either the mnemonic or symbol may be used.

3.10.2 Character String:

A character.string is a sequence of characters enclosed in quotes (").

Selecting part of a value.string can be performed by including any or all of the three following reserved characters in the character.string.

character	meaning
[	accept any leading characters or nulls.
]	accept any trailing characters or nulls.
^	accept any single character.

To use "[" in this special way it must be the first character in the character.string. To use "]" in this special way it must be the last character in the character.string.

The character "^" may occur anywhere in the string, and any number of times. Each case of "^" will match any single character in that location in the string.

These special "wild card" character meanings are not evaluated on elements that have EXTERNAL Operations defined, which are used as input conversions (see SMA:301).

3.10.3 ORing Value.Strings:
Several value.strings in succession are taken to be ORed; that is, if any one matches the referenced data from the item, the item passes this criterion.

3.10.4 ANDing Value.Strings:
The AND evaluation connective may be placed between value.strings. In this case, the referenced data must pass each of the tests stipulated by the relational connective-value string pair.

3.11 Sort.Criteria:
A sort criterion clause is made up of a sort activation connective and an associated data.definition.item specifier. The sort criterion applies to the sorting verbs; SORT, SSELECT, SREFORMAT, S-DUMP, SORT-ITEM, and SORT-LABEL.

If a sort.criteria is not specified on a sorting verb, the output of the command will be in the ascending order sorted sequence of the item-ids. Justification is determined by the File Definition Item (see SMA:301).

3.11.1 Sort Connectives:
The sort connectives for single-valued attributes are of the form:

BY element	- ascending order
BY-DSND element	- descending order

where "element" is a data.definition.item.

3.11.2 Exploding Sort Connectives:
The sort connectives for exploding multi-valued attributes, where each value is treated as an independent item by the sort.criteria, are of the form:

BY-EXP element {explosion.limiter}	- ascending order
BY-EXP-DSND element {explosion.limiter}	- descending order

where "element" is a data.definition.item and explosion.limiter is of value.string.critera form (see section 3.10).

3.11.3 Sort Evaluation:

The evaluation of a sort sequence criterion may be either alphabetic or numeric. It is considered alphabetic and left adjusted if the justification is not explictly declared to be 'right adjusted'. It will be evaluated from left to right.

The sort sequence is considered numeric if the justification is declared to be 'right adjusted'. The data will be evaluated in terms of numeric magnitude, including the sign. If value.string is alphanumeric, the numeric portions are sorted right-to-left, and the non-numeric portion is sorted left-to-right.

3.11.4 Multiple Key Sort:

Any number of sort criteria may be included in a statement. Each will contribute to the value used for sort purposes, with the first being used as the highest order sort value, and continues, in order, toward the end of the command, with the item-id always being used as the lowest-order sort value.

3.12 Output.Criteria:

Any data.definition.item not preceded by a selection connective or a sort connective is an output criterion.

The form of the output.criteria is:

{TOTAL} data.definition.item {print.limiter}

or

BREAK-ON data.definition.item {break.option.string}

The effect of an output.criteria is to emit the value.string generated by the data.definition.item into the output stream generated by the command.

If no output.criteria is specified and a set of default data.definition.items exist, they will be used for the output criteria. Default data.definition.items are those whose item.ids are sequential integers beginning with 1.

3.12.1 Print.Limiter Criteria:

Print.limiters are used with output.criteria to select certain values from multi-valued attributes for output. Output of values is limited to those values that meet specified criteria. Dependent values in associative data sets will be suppressed if the value they depend on is not output. A print.limiter criteria is of the same form as a value.string.criteria (see section 3.10).

3.12.2 BREAK-ON Connective:

The output.criteria may be preceded by the BREAK-ON connective. This causes the stipulated output.criteria values in successive items to be monitored for change. When an item is encountered which contains a different value from the previous item, a control break is said to have occurred. When a control break is detected, then special actions are taken to indicate the detection in the output, including the inclusion of the break.option.string.

The form of the break.option.string is:

"{{text}{'options'}}..."

The permissible BREAK-ON options are:

Options	Meaning
B	BREAK. Insert the value of the data.definition.item in the page heading.
D	DATA. Suppresses the break line entirely if there was only one detail line since the last control-break occurred.
L	LINE. Suppresses the blank line before the break data line. This option is ignored when the 'U' option (below) is used.
N	Reset the page number to one on this break.
P	PAGE. Cause a page break after this break line has been output.
R	ROLLOVER. Inhibit page break until all data associated with this break has been output.
U	UNDERLINE. Causes the underlining of all specified TOTAL fields.

3.12.2 BREAK-ON Connective (continued):

V	VALUE. Causes the value of the control break to be inserted at this point in the BREAK-ON line.
''	Two successive single quotes are used to insert a single quote mark in the text.

The control break process prints the leading and trailing text, if any, and the value of the data before the break if the V option is specified.

If there is neither text nor a V option specified, the default output is three asterisks (***).

3.12.3 Multiple Control Breaks:

Control breaks are hierarchically ordered, with the first control break criteria as the highest break level, and continuing toward the end of the command.

The use of control breaks assumes a sort sequence based on the data.definition.items specified as control breaks and in the same hierarchy.

There are at least 15 control break levels.

3.12.4 TOTAL Connective:

If an output.criteria is preceded by the TOTAL connective, then the data elements are summed for the specified items to be output.

There is one running total kept for each control break output.criteria, and one for the complete operation.

Totals are output for each output.criteria preceded by a TOTAL connective at each control break and at the end of the output.

Non-numeric data is taken to be zero for the totaling process.

3.12.5 GRAND-TOTAL Modifier:

The GRAND-TOTAL modifier may be used with the TOTALs and/or BREAK-ONs to specify special formatting on the grand-total line.

The GRAND-TOTAL connective form is:

```
GRAND-TOTAL {"text{'options'}text"}
```

where text is output at the completion of the command, and the 'options' enclosed in single quotes specify the following:

Option	Meaning
L	Line suppress. Suppress the line before the grand-total line.
P	Page break. Force a page break before the grand-total line is displayed.
U	Underline. Display a line of equal-signs (=) in the totaled output.criteria column before displaying the grand-total line.
''	Two successive single quotes are used to insert a single quote mark in the text.

The grand-total literal string will be displayed, left-justified, starting in the first column.

3.13 Modifiers:

Modifiers change the form of the output format. Note that several of the modifiers can alternatively be specified as Options.

The admissible modifiers are:

Mnemonic	Meaning
COL-HDR-SUPP	Suppress the default heading, the column headings, and the end-of-file message.
DBL-SPC	Place a blank line between items.
DET-SUPP	Suppress detail lines.
HDR-SUPP	Suppress the default heading and the end-of-file message. The column headings are not suppressed.

3.13 Modifiers (continued):

HEADING	Provide page heading format.
ID-SUPP	Suppress output of default item-id column.
FOOTING	Provide page footing format.
LPTR	Send the output to the spooler.
NOPAGE	Do not pause at the bottom of each page when displaying to a terminal.
TAPE	Acquire data from a T-DUMP tape.

3.13.1 Heading and Footing Modifiers:

The HEADING and FOOTING modifiers must be followed by a string surrounded by quotes ("), of the form:

```
"{{text}{'options'}}..."
```

Each group of options must be surrounded by single quotes ('). The effect is the text associated with the HEADING modifier will appear as the heading on each page of output, as operated on by the options, and the text associated with the FOOTING modifier will appear as the footing on each page of output, as operated on by the options.

3.13.2 Heading and Footing Options:

The allowable options for the HEADING and FOOTING modifiers are:

Option	Meaning
B	Break. Insert the value causing the control break if the "B" option has been specified in a BREAK-ON connective literal.
C	Center. Causes the HEADING or FOOTING line to be centered on the output page.
D	Date. Insert the current date, dd mmm yyyy, in the heading at this point.

3.13.2 Heading and Footing Options (continued):

F	File name. Insert the name of the file being LISTed or SORTed.
L	New line. Specifies a new line in the HEADING or FOOTING.
P	Page number. Insert the current page number, right justified, in a field of four blanks.
T	Time and Date. Insert the time and date, hh:mm:ss dd mmm yyyy.
''	Two successive single quotes are used to insert a single quote mark in the heading text.string.

3.14 Options:

The options must be last in the sentence, are enclosed in parentheses, and may be separated by commas. The right (closing) parentheses is optional.

The allowable options are:

Option	Meaning
B	Suppress initial terminal line-feed prior to output.
C	Refer to Modifier COL-HDR-SUPP
D	Refer to Modifier DET-SUPP
H	Refer to Modifier HDR-SUPP
I	Refer to Modifier ID-SUPP
N	Refer to Modifier NOPAGE
P	Refer to Modifier LPTR

3.15 Connective Synonyms:

A set of common synonyms are provided for the various connectives in the language as follows:

Synonym	Standard Connective
&	AND
EVERY	EACH
IF	WITH
WITHOUT	WITH NO
AFTER	GT
BEFORE	LT
NOT	NO
SUPP	HDR-SUPP
HEADER	HEADING
CAPTION	GRAND-TOTAL

3.16 Throw-away Connectives:

A set of common words are included that are not used by the language but may be included by the user to make the sentence more readable, which are as follows:

Throw-aways
!
A
AN
ARE
DATA
FILE
FOR
IN
ITEMS
OF
OR
THE

4.0 Verbs

4.1 LIST verb:

The SMA/Retrieval language verb, LIST, is used to generate a formatted output of selected items and attributes of a specified file.

The LIST verb has the following form:

```
LIST      {file.modifiers} file.name
          {item.list / item.id.selection.criteria}
          {USING {DICT} file.name}
          {data.definition.item.selection.criteria}
          {output.criteria{print.limiters}}
          {modifiers}{(option,...)}
```

See the appropriate sections for further details on each part of the sentence.

In the absence of an active item list, the output sequence of the LIST verb will be unordered. With an active item list, either explicit or implicit, the output sequence will be in the order given by the list.

4.2 SORT Verb

The SMA/Retrieval language verb, SORT, is used to generate a sorted and formatted output of selected items and attributes of a specified file.

The SORT verb has the following form:

```
SORT      {file.modifiers} file.name
          {item.list / item.id.selection.criteria}
          {USING {DICT} file.name}
          {data.definition.item.selection.criteria}
          {sort.criteria}
          {output.criteria{print.limiters}}
          {modifiers}{(option,...)}
```

See the appropriate sections for further details on each part of the sentence.

4.3 SELECT verb:

The SMA/Retrieval language verb, SELECT, creates a temporary implicit-list of the selected elements for later usage.

The SELECT verb has the following form:

```
SELECT   {file.modifiers} file.name
         {item.list / item.id.selection.criteria}
         {USING {DICT} file.name}
         {data.definition.item.selection.criteria}
         {output.criteria}
```

If there are no output.criteria specified, the item-ids are stored in a temporary implicit-list for use by the next verb as an implied 'item.list'. If the next verb is a SAVE-LIST verb, then the temporary implicit-list is saved. (See the section on SAVE-LIST).

If output.criteria are specified, the value of the specified attribute(s) will be saved in the implicit-list. Each value of a multi-valued attribute is treated as if it were in a single-valued attribute. The item-ids will not be saved in the select-list when output.criteria are specified.

The output from the SELECT verb is a temporary implicit-list and a message specifying the number of elements selected.

4.4 SSELECT verb:

The SMA/Retrieval language verb, SSELECT, creates a sorted temporary implicit-list of the selected elements from a file, for later usage.

The SSELECT verb has the following form:

```
SSELECT  {file.modifiers} file.name
         {item.list / item.id.selection.criteria}
         {USING {DICT} file.name}
         {data.definition.item.selection.criteria}
         {sort.criteria}
         {output.criteria}
```

The output of the SSELECT verb is the same as the SELECT verb, with the exception of the order.

4.5 COUNT Verb:

The SMA/Retrieval language verb, COUNT, will count the number of items in a file meeting the criteria as specified by the combination of item.list and/or selection.criteria.

The COUNT verb has the following form:

```
COUNT     {file.modifiers} file.name
          {item.list / item.id.selection.criteria}
          {USING {DICT} file.name}
          {data.definition.item.selection.criteria}
          {(option,...)}
```

A message is displayed showing the number of those items meeting the specifications of the item.list and/or the selection.criteria if present. If neither are specified, then the number displayed is the number of items in the specified file.

4.6 SUM Verb:

The SMA/Retrieval language verb, SUM, will generate the sum of the data elements of the items specified by the selection.criteria.

The SUM verb has the following form:

```
SUM       {file.modifiers} file.name
          {item.list / item.id.selection.criteria}
          {USING {DICT} file.name}
          {data.definition.item.selection.criteria}
          data.definition.item
          {(option,...)}
```

The output produced by the SUM verb includes the output title for the data.definition.item and the computed total.

4.7 STAT Verb:

The SMA/Retrieval language verb, STAT, generates a set of statistics for data elements in the items of a file.

The STAT verb has the following form:

```
STAT     {file.modifiers} file.name
         {item.list / item.id.selection.criteria}
         {USING {DICT} file.name}
         {data.definition.item.selection.criteria}
         data.definition.item
         {(option,...)}
```

The output of the STAT verb includes the output title for the data.definition.item, the generated sum of all data elements, the average of the data elements (total divided by count), and the number of items used in the generation of the statistics.

4.8 Reformat Verbs:

The SMA/Retrieval Language verbs, REFORMAT and SREFORMAT, are equivalent to the LIST and SORT verbs, except that the output is directed to another file or to tape, instead of a terminal or printer.

The REFORMAT and SREFORMAT verbs have the following forms:

```
REFORMAT    {file.modifiers} file.name
            {item.list / item.id.selection.criteria}
            {USING {DICT} file.name}
            {data.definition.item.selection.criteria}
            {output.criteria{print.limiters}}
            {modifiers}{(option,...)}

              .... and ....

SREFORMAT   {file.modifiers} file.name
            {item.list / item.id.selection.criteria}
            {USING {DICT} file.name}
            {data.definition.item.selection.criteria}
            {sort.criteria}
            {output.criteria{print.limiters}}
            {modifiers}{(option,...)}
```

After the REFORMAT or SREFORMAT sentence is entered, the system will prompt for:

FILE NAME:

The name of the file where the output is to be stored, or the word 'TAPE' if the output is to go to tape, must be entered.

4.8.1 Reformatting To Another File:

When reformatting into another file, the first value specified in the output.criteria is used as the item-id for the item, and the remaining values in the output.criteria are attributes in the item. Each item selected becomes an item in the new file.

4.8.2 Reformatting To The Same File:

When reformatting to the same file, the first value specified in the output.critera is used as the item-id for the item, and the remaining values in the output.criteria are attributes in the item. Each item selected becomes an additional item in the file. On a REFORMAT of a file onto itself, an implicit or explicit item list must be defined; otherwise, an infinite loop in which items are added to the file may occur.

4.8.3 Reformatting To Tape:

When reformatting a file to tape, the values specified in the output.criteria are concatenated together to form one tape record for each item that is selected. The record output is either truncated or padded at the end with nulls (hex '00's) to obtain a record the same length as specified by the last T-ATT verb.

A tape label which contains the file name, tape record length (in hex), the time and date, is written on the tape first, unless the HDR-SUPP or COL-HDR-SUPP modifiers or the options H or C are specified. Two End-Of-File marks (EOF's) terminate the file on tape. The item-id's will be displayed as the items are dumped unless the ID-SUPP modifier or the I option is specified.

4.8.4 Reformatting with an Exploding Sort Connective:

If an Exploding Sort Connective is used for the first data.definition.item specified in the output.criteria, then an item is created in the new file for each exploded entry. The remaining output.criteria attributes are repeated for each exploded entry.

4.9 Label Verbs:

The SMA/Retrieval Language verbs, LIST-LABEL and SORT-LABEL, are used for printing mailing labels or other special purpose listings.

Functionally, the LIST-LABEL and SORT-LABEL verbs are almost identical to the LIST and SORT verbs, except that the data associated with each item is grouped into a block by the LIST-LABEL and SORT-LABEL verbs, and several blocks can be placed across each page of a listing.

The LIST-LABEL and SORT-LABEL verbs have the following forms:

```
LIST-LABEL  {file.modifiers} file.name
            {item.list / item.id.selection.criteria}
            {USING {DICT} file.name}
            {data.definition.item.selection.criteria}
            {output.criteria{print.limiters}}
            {modifiers}{(option,...)}

              .... and ....

SORT-LABEL  {file.modifiers} file.name
            {item.list / item.id.selection.criteria}
            {USING {DICT} file.name}
            {data.definition.item.selection.criteria}
            {sort.criteria}
            {output.criteria{print.limiters}}
            {modifiers}{(option,...)}
```

After the sentence is entered, an additional set of parameters is prompted for with a question mark (?) until a null line of data ([CR]) is entered. The first additional parameter must be entered and has the following format:

```
?count,row,skip,indent,size,space{,C}
```

These parameters determine the arrangement of attribute values into blocks and are entered in the following format:

Parameters	Meaning
count	The number of items (labels) across each page.
rows	The number of lines printed for each label (height of each label, in ...rows)
skip	The number of lines to skip between labels (vertical spacing between labels, in rows)

4.9 Label Verbs (continued):

Parameters	Meaning
indent	The number of spaces to indent the the data from the left margin.
size	The maximum width permitted for the data associated with each attribute name (width of each label, in columns)
space	The number of horizontal spaces between labels in columns
C	Optional; if present, specifies that null attributes are not to be printed. If the C is not specified, null values will be printed as all blanks.

The values must conform to the following range:

(count * size) + ((count-1) * space) + indent <= (page width)

where 'page width' is the number defined for the output device (terminal or spooler).

Otherwise the system will respond with an error message indicating that an invalid numeric parameter was entered.

The normal non-columnar list heading (page number, time and date) will print on the top of each page unless suppressed by the COL-HDR-SUPP modifier or (C) option. If headings are suppressed, pagination and all top-of-forms are suppressed, then a continuous forms structure without page breaks is produced.

A set of row header data lines will be requested, immediately following the first parameter request, if the 'indent' parameter is non-zero. The parameter 'rows' specifies how many row headers will be requested because one row header is printed for each row of the label. The row headers will be printed in the 'indent' area of the left-hand margin. Null headers may be specified by entering null lines ([CR]) to the header data requests.

4.10 Item Listing Verbs:

The SMA/Retrieval language verbs, LIST-ITEM and SORT-ITEM, copy a complete item to the terminal or spooler.

The LIST-ITEM and SORT-ITEM verbs have the following forms:

```
LIST-ITEM    {file.modifiers} file.name
             {item.list / item.id.selection.criteria}
             {USING {DICT} file.name}
             {data.definition.item.selection.criteria}
             {modifiers}{(option,...)}

               .... and ....

SORT-ITEM    {file.modifiers} file.name
             {item.list / item.id.selection.criteria}
             {USING {DICT} file.name}
             {data.definition.item.selection.criteria}
             {sort.criteria}
             {modifiers}{(option,...)}
```

The data from each attribute in the item, with a three digit number in the left margin, is copied to the terminal or spooler.

The permissible LIST-ITEM and SORT-ITEM verb options are:

Option	Meaning
F	Causes a Form-Feed for each item. Starts a new page for each item.
N	Inhibits the pause at the end of each page when the output is to the terminal (NOPAGE).
P	Sends the output to the spooler (LPTR).
S	Suppresses the three digit line number in the left margin.

The equivalent modifiers for the options are shown in parentheses above.

4.11 FILE-TEST Verb:

The SMA/Retrieval language verb, FILE-TEST, provides a means of generating statistics for a specified file, or testing a new configuration for a specified file.

The FILE-TEST verb has the following form:

```
FILE-TEST   {file.modifiers} file.name
            {item.list / item.id.selection.criteria}
            {USING {DICT} file.name}
            {data.definition.item.selection.criteria}
            {modifiers}{(option,...)}
```

where 'file.name' is the name of the file for which statistics are to be generated.

After the FILE-TEST sentence has been entered, the system may prompt for one or more parameters which may be used to define an alternate configuration for the file. If a null return is supplied to the prompt, then the current file configuration parameters will be used.

The statistics that are generated for the selected items will include; the count of items, the total number of bytes in all the items, and the average number of bytes per item. In addition there may be statistics relevant to the structure of the specified file. This structural information may be presented in different ways, as in the case of a hashing structure with the histogram, the average number of items per group and standard deviation, and the average number of bytes per group.

If no 'item.list' or 'select.criteria' are specified, then all the items in the file will be used for generating the statistics.

The option (S) causes the detail information to be suppressed, and only the statistical data to be printed.

4.12 CHECK-SUM Verb:

The CHECK-SUM verb generates a checksum for file items.

The CHECK-SUM verb has the following form:

```
CHECK-SUM    {file.modifiers} file.name
             {item.list / item.id.selection.criteria}
             {USING {DICT} file.name}
             {data.definition.item.selection.criteria}
             {data.definition.item}
             {(option,...)}
```

The checksum is calculated as an arithmetic total, disregarding overflow, of all the bytes in the selected data elements. If a data.definition.item is specified, then only the values of the specified data elements will be used in the calculation.

The result is presented in a message indicating the number of items checked and the checksum calculated.

4.13 Tape Verbs And The Tape Modifier:

The SMA/Retrieval language verbs, T-DUMP and T-LOAD, provide the facility to write the items of a specified file to tape, or to load items into a specified file from a previously generated T-DUMP tape. The TAPE modifier may be used with other SMA/Retrieval language verbs to access data from a T-DUMP tape.

4.13.1 Tape Dumping Verbs:

The T-DUMP verb will dump the specified items from a specified file to tape. The S-DUMP verb will also dump specified items from a file to tape, except that the items will be sorted before they are dumped. The tape drive must be first attached to the users account via the T-ATT command. The T-ATT command is also used to set the record length to be used by the T-DUMP or S-DUMP verb.

The T-DUMP and S-DUMP verbs has the following form:

```
T-DUMP      {file.modifiers} file.name
            {item.list / item.id.selection.criteria}
            {USING {DICT} file.name}
            {data.definition.item.selection.criteria}
            {HEADING "text"}
            {modifiers}{(option,...)}

and

S-DUMP      {file.modifiers} file.name
            {item.list / item.id.selection.criteria}
            {USING {DICT} file.name}
            {data.definition.item.selection.criteria}
            {HEADING "text"}
            {sort.criteria}
            {modifiers}{(option,...)}
```

T-DUMP and S-DUMP cause a standard tape label to be written on the tape. If the optional HEADING modifier and 'text' are specified, the 'text' is added to the label. See SMA:201 the SMA/Data Interchange Standard.

If the optional file.modifier 'DICT' is specified, the dictionary section of the specified file will be dumped with the exception of File Definition Items. A File Mark indicator is written on the tape at the end of the dump.

The HDR-SUPP modifier or (H) option may be used to suppress the writing of the tape label.

The ID-SUPP modifier or (I) option may be used to suppress the listing of item-ids that are dumped.

A message is displayed when the dump is finished indicating the number of specified items dumped to tape.

4.13.2 T-LOAD Verb:

The T-LOAD verb loads the specified items into a specified file. The tape must be attached before the T-LOAD verb is used.

The T-LOAD verb has the following form:

```
T-LOAD      {file.modifiers} file.name
            {item.list / item.id.selection.criteria}
            {USING {DICT} file.name}
            {data.definition.item.selection.criteria}
            {modifiers}{(option,...)}
```

The T-LOAD verb will read the SMA standard tape label if present and setup the tape record length from the label. If the tape is unlabeled or has a non-standard label, the record length must be set using the T-ATT verb.

The items, as restricted by the item.list or the selection.criteria, are loaded into the specified file if they do not exist in the file. The (O) option will permit the overwriting of items that exist in the file. If the item exists in the file and the (O) option was not specified, a message is displayed indicating the item-id of the item that exists on the file.

Any items existing in the file and not existing on the tape will be maintained.

Specifying the ID-SUPP modifier or the (I) option will suppress the listing of the item-ids during the loading of the specified file.

A message which indicates the number of items loaded will be displayed when the specified file has been loaded.

4.13.3 TAPE Modifier:

The TAPE modifier can be used to read data from a T-DUMP tape rather than the data portion of the specified file. The TAPE modifier can only be used with the verbs: LIST, LIST-LABEL, LIST-ITEM, SUM, STAT, FILE-TEST, or COUNT. The dictionary of the specified file will be used for the specified dictionary.definition.items.

4.14 List File Handling Verbs:

The SMA/Retrieval language provides a set of verbs for the saving, editing, copying, retrieving, forming, and deleting selected item.lists.

4.14.1 SAVE-LIST Verb:

The verb, SAVE-LIST, makes a stored item.list from a temporary implicit-list produced by the SELECT, SSELECT, and FORM-LIST verbs.

The SAVE-LIST verb has the following form:

```
SAVE-LIST {{DICT} file.name} list.name
```

The SAVE-LIST verb saves the temporary implicit-list and adds or updates the pointer to the list in the file POINTER-FILE, if file.name is not specified, with an item-id of list.name.

If file.name is specified, the list's pointer will be added or updated in the specified file. The file definition item of the specified file must declare that the file can hold indirect data pointers.

An existing stored list with the same name will be overlaid by the newly stored list.

The SAVE-LIST verb displays a message showing the list.name and the number of frames used to store the list.

The SAVE-LIST verb must be issued immediately after the creation of a temporary implicit-list in order to create a stored list.

4.14.2 GET-LIST Verb:

The verb, GET-LIST, retrieves a previously stored list and forms a temporary implicit-list.

The GET-LIST verb has the following form:

```
GET-LIST {{DICT} file.name} list.name
```

The list.name specifies which stored list is to be retrieved. If file.name is specified, the list is retrieved from the specified file. The POINTER-FILE is used if file.name is not specified.

If the specified item 'list.name' does not exist, a message indicating such is produced. If the item is found, a message is displayed showing the number of entries in the list and the temporary implicit-list is available for use.

4.14.3 DELETE-LIST Verb:
The verb, DELETE-LIST, deletes a stored item.list.

The DELETE-LIST verb has the following form:

```
DELETE-LIST {{DICT} file.name} list.name
```

The list.name specifies the stored list to be deleted. If the item does not exist, a message indicating such is produced.

If the item is found, a message is displayed stating that the list was deleted.

The POINTER-FILE is used if file.name is not specified.

4.14.4 FORM-LIST Verb:
The verb, FORM-LIST, will generate a temporary implicit-list from attribute(s) within an item or items in a file.

The FORM-LIST verb has the following form:

```
FORM-LIST {DICT} file.name {item.list} {(n)}
```

where the data is taken from the item(s) in the specified file. The item.list can be implicit, explicit, or an asterisk (*) specifying all items. All data from the items are stored in a temporary implicit-list unless the optional (n) specification is used; in which case, only data from the n-th attribute of each item is used. Multi-values or sub-values are stored as separate elements in the implicit-list.

A message indicating the number of list entries formed is displayed at the conclusion of the process.

THIS IS THE LAST PAGE

✓ 6/29/88

SMA STANDARD

SMA/PROC Language Specification

SMA: 501

March 1987

SPECTRUM MANUFACTURERS ASSOCIATION

sma

SMA/PROC LANGUAGE SPECIFICATION - DRAFT 2.5

-- 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.

Some material contained herein is designated as proprietary by individual member companies of SMA listed below. Any unauthorized use of such proprietary information is prohibited.

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/PROC language. This language, provided by most Spectrum Manufacturers Association member systems, is used primarily for job control. It provides a method of linking individual jobs together as a stored procedure. This document is intended to serve as a guide to the preparation of PROCs 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 individuls and organizations for their contributions in the preparation of this document:

I. Sandler,	CIE Systems
H. Eggers,	Mcdonnell Douglas Computer Systems Co.
J. Timmons,	Laguna Software & Consulting, Inc.
C. Wilson,	General Automation, Inc.
C. Saunders,	Fujitsu Microsystems of America, Inc.

CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

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 stored procedures that can be moved from one system to another with maximum portability.

1.2 Inclusions: This document includes all the commonly available statements with syntactical representations to clearly define the usage permitted with each. It also includes sufficient "run time" considerations to meet its objective of inter-machine portability.

1.3 Exclusions: In this version of this document, the syntactical aspects of the language are addressed, together with some common "run time" considerations. There are 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. Although many systems support 'user' exits in the language, they are not considered in the scope of this specification.

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 and brackets are explicit within the language, and must be considered part of the statement. The use of double quotes or single quotes is also specific in the language, and must be considered part of the statement.

2.2 Structural Terms:

Item-Id - A character string (maximum 48 characters) which uniquely identifies an item within a file. The item-id may include any characters except system delimiters, however the use of blanks, single quotes, quotes, commas, and backslash characters may require special considerations.

Item Body - The variable length character structure which makes up the information content of the item. It is composed of any number of attributes, values, and subvalues. A segment mark is used to mark the end of the structure.

IDP - Indirect Data Pointer: Special type of item body that locates the data stored separately from the item body. Used to store non-character structures such as programs, as well as other extended structures.

SM - Segment Mark: Delimiter character used to mark the end of an item body. It is represented graphically by an underscore. A segment mark may not be included within data.

AM - Attribute Mark: Delimiter character used to mark the end of an attribute. It is represented graphically by an up-arrow. The last AM in an item is implied by the presence of a segment mark.

VM - Value Mark: Delimiter character used to mark the end of a value. It is represented graphically by a right bracket. The last VM in an attribute is implied by an attribute or segment mark.

2.2 Structural Terms (continued):

SVM - Subvalue Mark: Delimiter character used to mark the end of a subvalue. It is represented graphically by a backslash. The last SVM in a value is implied by a value, attribute, or segment mark.

BM - Buffer Mark: Delimiter character used to mark the beginning or ending of special character strings. It is represented graphically by a left bracket.

AMC - Attribute Mark Count: The positional count, from 1, that locates a specific attribute within the body of an item. An AMC value of zero is used to locate the item-id.

VMC - Value Mark Count: The positional count, from 1, that locates a specific value within a multivalued attribute.

SVMC - Subvalue Mark Count: The positional count, from 1 that locates a specific subvalue within a multisubvalued structure.

3.0 Overview:

3.1 Concepts:

PROC provides the applications programmer a means to catalog a sequence of operations which can be invoked from the terminal by a one word command. Any operation that can be executed by the Terminal Control Language (TCL) can be performed in a PROC.

3.2 Execution:

A PROC stored as an item in the user's master dictionary is executed by typing in at the keyboard the name of the PROC, followed by any parameters, followed by a carriage-return.

3.3 Structure:

A PROC is stored as an item in a file. The first attribute of the PROC is always the two character literal "PQ". All subsequent lines of the PROC contain PROC statements which serve to generate TCL commands, pass parameters to other processes, control branching within the PROC, or manipulate the buffers used by the PROC processor. PROC statements consist of an optional unsigned integer label (used to uniquely identify its associated command for purposes of branching or looping within the PROC) followed by a space, and a one or two character command, together with optional arguments. PROC statements are executed interpretively by the PROC processor. Once a PROC is invoked, it remains in control until it is exited, even though it may temporarily relinquish control to another processor to execute a specific command.

Only one PROC statement may be defined per line. Although the IF statements appear to allow more than one, the additional statements are part of the IF structure. A PROC statement, which does not have a label, must not contain leading spaces.

3.4 PROC Buffers:

The PROC processor uses four buffers, two for input and two for output. These are known as the Primary Input Buffer, Secondary Input Buffer, Primary Output Buffer and Secondary Output Buffer respectively.

The Primary Input Buffer initially contains the data which invoked the PROC.

The Secondary Input Buffer is a volatile temporary workspace, usually used for temporary storage of input data. It is used to receive the error message number or numbers resulting from the prior TCL command.

3.4 PROC Buffers (continued):

The Primary Output Buffer is initally empty. It stores the command which is normally passed to the TCL processor for execution.

The Secondary Output Buffer (Stack) is initially empty. It stores any parameter strings required by the command stored in the Primary Output Buffer. Each request for terminal input by the invoked TCL command is satisfied by the parameter set from the Secondary Output Buffer. In the event that the called process requires more parameter strings than are present in this buffer, the data is requested from the terminal from that point onwards. Note that each parameter string in the Secondary Output Buffer must be explicitly terminated, with the exception of the last string, where the terminator is assumed if it does not explicitly exist.

3.5 Parameter Manipulation:

Moving data between the various PROC buffers is done in terms of "parameters". A parameter is defined as a string of characters (residing in one of the buffers), which is surrounded by blanks, single quotes, or double quotes. Missing parameters are either beyond the buffer or are represented by a backslash.

3.6 Active Buffer Pointer:

Two pointers are maintained that identify which buffer is active, the primary or the secondary. The active buffer pointer for the input buffers is initially set to the primary input buffer and the active buffer pointer for the output buffers is also initially set to the primary output buffer.

3.7 Parameter Pointers:

To keep track of the parameters in the buffers, there is an input pointer and an output pointer which maintain the current position in the currently active buffers. The parameter pointers are initially set to the first parameter in each of the two buffers.

4.0 Statement Syntax Definitions:

4.1 A **Move Parameter:** The general form of this command moves one parameter from the current input buffer to the current output buffer, starting at the current pointer positions of those buffers. This command may use one or more of the following optional parameters:

A{s}{n}{,m}

"s" is a surround character which can be any non-alphabetic, non-numeric character. Surround characters only apply to the primary output buffer. They are ignored when transferring parameters to the secondary output buffer. If no surround character is specified when transferring a string to the primary output buffer, a space is used as the default surround character.

Multiple elements within a parameter may be constructed by separating the elements with semi-colons. Upon recognition of the semi-colon, the previous element will be completed by the designated surround character and the following element will be opened by the surround character.

Frequently used versions of this parameter are:

A' will surround the data with single quotes (') when it moves it to the primary output buffer.

A" will surround the data with double quotes (") when it moves it to the primary output buffer.

A\ will use no surround characters when it moves the data to the primary output buffer.

"n" and "m" must be unsigned integers, and have different meanings, depending on whether or not they appear singly or together.

An will select the current input buffer and reposition the input buffer pointer to the start of parameter "n" prior to transfering the parameter.

4.1 A **Move Parameter (continued):**

A,m will move "m" characters (or until the end of input buffer is reached if that is less) from the input buffer to the output buffer. This will result in more than one parameter being moved if "m" is larger than the length of the parameter currently located by the pointer.

An,m will move "m" characters from the input buffer to the output buffer starting at parameter "n".

A(n,m) will move "m" characters from the input buffer to the output buffer starting at character "n". Thus A(1,9999) will transfer the entire input buffer to the output buffer (assuming there are 9999 or less characters in the input buffer).

After execution both the input and output pointers are left pointing one character after the last character transferred.

4.2 B **Backup The Current Input Pointer:** This command will backup the current input buffer pointer to the start of the previous parameter unless it is already at the start of the buffer when it will have no effect.

4.3 BO **Backup The Current Output Pointer:** This command will backup the current output pointer to the start of the previous parameter unless it is already at the start of the buffer when it will have no effect.

4.4 C **Comment:** Any text following this command is treated as a comment and is ignored by the PROC processor.

4.5 D **Display Buffers:** The general form of this command displays on the terminal parameters from the current input buffer, without affecting the position of any of the buffer pointers. This command may take one or more of the following options:

D{n}{,m}{+}

"n" and "m" must be unsigned integers, and have different meanings, depending on whether or not they appear singly or together. If the "D" command is terminated with a plus sign, no carriage-return/line-feed is output after the display.

D will display the current parameter.

4.5 D Display Buffers (continued):

Dn will display parameter "n" of the current input buffer rather than the current input parameter. The case of n=Ø is a special case which displays the entire input buffer.

D,m will display "m" characters, starting at the current input buffer position. This may result in more than one parameter being displayed.

Dn,m will display "m" characters, starting at parameter "n" of the input buffer.

D(n,m) will display "m" characters, starting at column "n" of the input buffer. Thus D(1,9999) will display the entire input buffer (assuming there are 9999 or less characters in the input buffer).

4.6 F Forward The Current Input Pointer: This command will move the current input buffer pointer forward to the start of the next parameter unless it is already at the end of the buffer when it will have no effect.

4.7 G Go To A Specific Label: This command will transfer control within the PROC to the line with the specified integer label. The general form of this command is:

```
G label
GO label
G An
GO An
```

The form "G An" or "GO An" may be used to go to a label supplied as a parameter in the primary input buffer. If the label supplied does not exist in the PROC, then execution will continue with the line following the Go command.

4.8 H Place Text In Output Buffer: This command will cause the text immediately following the "H" (including any blanks) to be copied to the current output buffer, starting at the position pointed to by the output pointer for that buffer. Note that each parameter string in the Secondary Output Buffer must be explicitly terminated with a less than (<) character, with the exception of the last where the terminator is assumed. The use of two less than characters, as termination, provides for line continuation of the secondary buffer.

4.9 IF Conditional Test: This command provides a method of testing and validating parameters in the current input buffer.

The following variants of this command refer to the forms of the "A" command documented in section 4.1. Note that only those forms of the "A" command which determine the position of the input buffer pointer (A{n}{,m}) are acceptable here, and that the input pointer is not changed by the "IF" command.

There are five distinct forms of this statement:

4.9.1 IF Conditional Test - Form 1:

```
IF A{n}{,m} PROC.cmd
IF #A{n}{,m} PROC.cmd
```

This form will test if there are any characters in the string specified by A{n}{,m}, and execute PROC.cmd if the test is satisfied.

Note that missing parameters, which are represented by a backslash, will test true.

4.9.2 IF Conditional Test - Form 2:

```
IF A{n}{,m} operator string PROC.cmd
```

This form will test the string specified by A{n}{,m} against the specified string of characters, and execute PROC.cmd if the test is satisfied.

Valid operators are:

```
 =   Test for equal values
 #   Test for unequal values
 <   Test if parameter is less than string
 [   Test if parameter is less than or equals string
 >   Test if parameter is greater than string
 ]   Test if parameter is greater or equals string
```

Note that the above tests are on a character by character basis only. Thus numeric strings with leading zeros will not test the same as numerics without the leading zeroes.

4.9.3 IF Conditional Test - Form 3:

IF A{n}{,m} operator (format.string) PROC.cmd

This form will test the string specified by A{n}{,m} against the specified format.string, and execute correct format.

The format.string can be any combination of:

nN - test for "n" numeric characters
nA - test for "n" alphabetic characters
nX - test for "n" characters

any other characters found in the string will be assumed to be literals and will be tested for as such. If "n" is zero, all characters which satisfy the test will be skipped. Literals appearing in any of the forms listed above may be enclosed in single quotes to insure they will be processed as literals.

4.9.4 IF Conditional Test - Form 4:

IF E PROC.cmd
IF #E PROC.cmd
IF E=value PROC.cmd
IF E#value PROC.cmd

This form will test the error message number returned by the immediately previous operation and execute PROC.cmd if the test is satisfied.

4.9.5 IF Conditional Test - Form 5:

IF S PROC.cmd
IF #S PROC.cmd

This form will test if a select list is currently active and execute PROC.cmd if the test is satisfied.

4.1Ø IH **Place Text In Input Buffer:** This command causes the line of text immediately following the "IH" (including any blanks) to replace the current parameter in the current input buffer. The input buffer pointer continues to point to the beginning of the replaced string after execution of the command. The form "IH\" replaces the current parameter with a backslash which is used, by convention, to indicate a missing parameter.

4.11 IP **Input Data To Primary Input Buffer:** This command causes the PROC processor to prompt for input at the terminal. Data entered by the user replaces the current parameter of the primary input buffer. If the pointer is at the end of the input buffer, the data is appended to the buffer. After execution of this command, the pointer points to the start of the newly entered parameter. The general form of the command is:

IP{B}{p}

The "B" option with this command converts embedded spaces in the input data into backslashes (\).

The "p" option, if used, changes the default prompt character of colon (:) to the character specified.

4.12 IS **Input Data To Secondary Input Buffer:** This command causes the PROC processor to prompt for input at the terminal. This buffer is affected by many operations and must be used carefully. The general form of this command is:

IS{p}

The "p" option, if used, changes the default prompt character of colon (:) to the character specified.

4.13 IT **Input Tape Label to Primary Input Buffer:** This command causes the PROC processor to read the tape label from the attached tape and copy the label into the cleared primary input buffer. The general form of this command is:

IT

If no tape label exists, then both input buffers are cleared as with the "RI" command.

4.14 O **Output Text To Screen:** The string of characters immediately following the command are output to the terminal followed by a carriage-return and line-feed. The carriage-return and line-feed are suppressed if the last character of the string is a plus sign (+).

4.15 P **Perform Current Command:** This command executes the current contents of the primary output buffer as though it had been entered at the terminal. After execution PROC regains control at the statement immediately following the "P" command. If the command being executed requires further operator input, it will obtain it from the secondary output buffer, and only prompt for input at the terminal if the secondary output buffer has been exhausted. Excess parameters in the secondary output buffer are discarded. This command has the following general form:

P{H}{P}{W}{X}

H will hush all output to the terminal for the command being executed.

P will display the current contents of the PROC output buffers before executing the command.

W will display the current contents of the PROC output buffers and wait for input. If the letter "S" is entered, the current command is skipped. If the letter "X" is entered PROC execution is aborted. A carriage return or the letter "Y" input will cause execution to continue normally.

X will cause the system to not return to the PROC after it has processed the specified command.

Note: The "P" command causes the input buffer pointer to be positioned at the start of the primary input buffer. The secondary input buffer will contain error message numbers output by the processed command. The output buffers will be initialized.

If the command in the output buffer is a PROC, then the newly activated PROC will be entered with the primary input containing the character string from the output buffer of the PROC that issued the "P" command. All other buffers are initialized. Control will NOT return to the original PROC.

4.16 RI **Reset Input Buffer:** This command resets both the primary and secondary input buffers to an empty or null condition. The general form of the command is:

RI{n}

"n", if specified, will reset the primary input buffer so that only parameters 1 through "n" remain.

After executing this command, the primary input buffer is always selected as the current input buffer.

4.17 RO **Reset Output Buffer:** This command resets both the primary and secondary output buffers to an empty or null condition. After executing this command the primary output buffer is always selected as the current output buffer.

4.18 S **Set Position Of Input Pointer:** This command selects the primary input buffer as the current input buffer and positions the input pointer at the start of the specified parameter in the current input buffer. The general form of the command is:

S{n}

"n" positions the pointer at the start of the "n"th parameter. Thus both "SØ" and "S1" position the pointer at the start of the buffer. If there is no "n"th parameter, sufficient backslash placeholders are created to position the pointer correctly.

4.19 SP **Select Primary Input Buffer:** This command makes the primary input buffer the active input buffer, and sets the input parameter pointer to the start of the buffer.

4.2Ø SS **Select Secondary Input Buffer:** This command makes the secondary input buffer the active input buffer, and sets the input parameter pointer to the start of the buffer.

4.21 STON **Stack On:** Select the secondary output buffer (the stack) as the active output buffer. "ST ON" is also accepted as this command.

4.22 STOFF **Stack Off:** Select the primary output buffer as the active output buffer. "ST OFF" is also accepted as this command.

4.23 T **Formatted Terminal Output:** This command is used to output strings of data at specific positions on the terminal. It is followed by a variable number of parameters separated by commas. The general form of the command is:

T parameter {,parameter}...

Each parameter can be any one of the following:

text A literal string enclosed in single or double quotes.

B Causes the bell to be sounded on the terminal.

C Clears the terminal's screen.

In Causes the system to output the character whose value is given by the decimal number immediately following the "I".

Xn Causes the system to output the character whose value is given by the hexadecimal number immediately following the "X".

(x,y) Causes the terminal to position its cursor at column "x" of row "y" on the screen using the systemwide standard cursor routine.

(-v) Causes the terminal to perform specific operations as defined in SMA:101 (SMA/BASIC Language Specification).

4.24 X **Exit From The PROC:** This command causes the current PROC to be terminated. If it was called by a prior PROC as a subroutine, control is returned to the calling PROC, otherwise PROC is terminated and control returns to TCL. If the "X" is followed by a character string, that string is displayed on the terminal when the command is executed.

4.25 (**Transfer Control Jump:** This command transfers control to a different PROC. The general form of the command is:

({DICT} filename {item-id}){label}

where: If the optional "item-id" is not specified, it is taken to be the current parameter in the input buffer. If the optional "label" is specified, then execution begins at that line within the new PROC. If the "label" is not found within the destination PROC, then an error message is presented and control returned to TCL.

NOTE that transferring control does not affect the contents of any of the input or output buffers.

4.26 [**Transfer Control Subroutine:** This command is very similar to a transfer control jump. It differs in that when the called PROC is exited, control returns to the next line of the calling PROC. The general form of the command is:

[{{DICT} filename {item-id}}]{label}

where: If the optional "item-id" is not specified, it is taken to be the current parameter in the input buffer. If the optional "label" is specified, then execution begins at that line within the new PROC. If the "label" is not found within the destination PROC, then an error message is presented and control returned to TCL.

4.27 +n **Add To Current Parameter:** This command adds the integer "n" to the current value of the current input parameter. This command only works if the current input parameter contains an integer value, and only gives the correct result if the result of the addition does not generate a number containing more characters than the original value.

4.28 -n **Subtract From Current Parameter:** This command subtracts the integer "n" from the current value of the current input parameter. This command only works if the current input parameter contains an integer value, and only gives the correct result if the result of the subtraction does not generate a number containing more characters than the original value.

THIS IS THE LAST PAGE

✓ PPG 4
6/17/88

SMA STANDARD

RUNOFF Language Specification

SMA: 601
January 1988

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.

Some material contained herein is designated as proprietary by individual member companies of SMA listed below. Any unauthorized use of such proprietary information is prohibited.

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

SMA:601
RUNOFF LANGUAGE SPECIFICATION - DRAFT 2.0

Foreword: This document establishes a standard for the use of the RUNOFF language and processor to prepare textual material on-line. The RUNOFF language, provided by Spectrum Manufacturers Association member systems, is used primarily for generating form letters and text documentation. This document is meant to serve as a guide to the prepartion of RUNOFF items that can be moved from one SMA system to another. For 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 to the preparation of this document:

C. Saunders, Fujitsu Microsystems Of America, Inc.
J. Timmons, Data Cache, Inc.
J. Treankler, JET Software, Inc.

CONTENTS

CONTENTS

1.0 Scope

1.1 Inclusions: This document includes all commands and features common to all SMA systems with syntactic respresentations to clearly define the usage permitted with each. It also includes sufficient "run time" considerations to meet the objective of inter-system portability.

1.2 Exclusions: Excluded from this standard are issues of support of statements during the run time process by other SMA system processors, such as the output spooler and CRT handlers. Also excluded is any discussion of how RUNOFF source items are created.

2.0 Definitions

2.1 Nomenclature: Within this document, capitalized words represent tokens within the RUNOFF language and must be included as shown. RUNOFF commands will be recognized without regard to case. For example, both ".NCS" and ".ncs" will turn off the Capitalize Sentence mode.

Terms in lower case refer to parameters which must be supplied as part of the RUNOFF command.

The use of quotes (") and single quotes (') is required in the forms shown below.

The use of braces ({ }) means the included string is optional.

The use of ellipsis (...) means the preceding information can be repeated.

The term "printed" refers to output to the specified device which may be a terminal, an auxilary device connected to the terminal, the system spooler or to the tape device.

3.0 Overview

3.1 Concepts: RUNOFF provides users the means to output text documents that have been prepared on a SMA computer system. The text can be entered via any other process, for example the SMA/EDITOR. As noted before, the entry of text is not discussed in this document, as RUNOFF is a text output formatter. It takes text items with RUNOFF commands embedded within the text and generates output. Features of RUNOFF allow the generation of form letters with data inserted by RUNOFF from an application database as well as the creation of books and pamphlets, with indexes and tables of contents maintained and printed automatically.

3.2 RUNOFF Item Structure: Input to the RUNOFF processor is contained in standard SMA items. All attributes within the item are considered to be text, except attributes that begin with a period (.) which are RUNOFF command lines. RUNOFF command lines contain one or more RUNOFF commands, each command prefaced with a period. Some RUNOFF commands also use the following text line for special situations, such as headings and footings.

3.3 Process Initiation: The RUNOFF Processor is invoked from TCL with the following statement:

```
RUNOFF {DICT} filename {itemlist} {(options)}
```

where the valid options are:

- n — Any positive integer number which specifies the number of times to repetitively print character(s) which are printed in boldface mode.
- C — Suppresses linking to other items via the CHAIN and READ commands.
- I — Outputs each source item name before generating text output.
- J — Suppress functioning of the highlighting mode.
- N — Suppresses the pause at end of page when output is directed to the terminal.
- P — Directs output to the system spooler.
- S — Suppress functioning of the Boldface and Underline modes.
- U — Specifies that all lower case characters will be converted to upper case during printing.

The "itemlist" specification designates an explicit itemlist, which is made up of source RUNOFF item ID(s) separated with blanks. The item id(s) do not have to be enclosed in quotes or single quotes.

If there is no explicit itemlist, there must be an implicit itemlist, generated by a previous SELECT, SSELECT, FORM-LIST, or GET-LIST verb. See the SMA/Retrieval Language Specification for information on these verbs.

With one exception, the explicit itemlist over-rides the implicit itemlist. The exception is when the READNEXT RUNOFF command is used. When used, READNEXT requires both the implicit and explicit itemlists. Only the first item in the explicit itemlist is used.

3.4 Execution: Execution begins with the first item specified and continues until all items in the itemlist are output. Within each item execution begins with the first attribute and continues with each succeeding attribute until the end of item is reached.

RUNOFF begins processing each line by testing the first character of the line for a period, ".", which identifies a RUNOFF command line. Each RUNOFF command on that line is processed as specified by the command definitions in chapter 5.

If RUNOFF is in Fill Mode, the text line is parsed into words (separated by spaces) and these words are placed into a temporary buffer whose length is defined by the LINE LENGTH command (and, in the case of a Paragraph Break, by Paragraph Indent.) When RUNOFF is unable to place a complete word in the buffer, the buffer is printed and then emptied. Processing continues with the word which caused the overflow.

If RUNOFF is not in Fill Mode, the source text line is printed as it appears in the source item.

Before a line is printed, RUNOFF will adjust the line based on the Box, Highlight and Justify Modes. The line is prefaced with the number of spaces indicated by a combination of Left Margin, Temporary Indent, Indent, Offset Indent, and Paragraph Indent. It is then sent to be printed.

In the special case that nothing has been printed yet, a Page Break is generated before the line is printed.

After each line is printed, the Line Spacing Counter is checked. If the Line Spacing Counter is greater than one, additional blank lines are printed so that the blank lines plus the text line equals the Line Spacing Counter. The Current Line Counter is incremented by the amount of the Line Spacing Counter. If the

Current Line Counter plus the number of lines to be generated by the Footing Tag is greater exceeds the Paper Length, a Page Break is generated.

4.0 Processing Mechanisms

RUNOFF can manipulate text in a variety of ways. To accomplish some of these effects, RUNOFF requires the use of mechanisms to remember such things as the number of lines already printed on the current page and whether text should be printed with a justified right margin.

These mechanisms are described below. They can be broken down into three catagories: A) Tags and Counters, which store user specified information or accumulated results. B) Output Breaks, which affect the format of output. C) Modes, which flag whether RUNOFF is to take specified action automatically and repeatedly.

4.1 Footing and Heading Tags: The Footing and Heading Commands use the subsequent line of text, known as the tag line, as what to print at the top or bottom of each page. The tag line is text with options embedded within it. Options are specified within single quotes (') and may be any combination of the following:

- C Centers the tag output line between left and right margins. If the "L" option is being used, then the "C" option must be repeated for each of the output lines that is to be centered.
- D Prints the current date.
- F Prints the source file name.
- I Prints the source item id.
- L Prints a carriage return and linefeed.
- P Prints the current page number right justified in a field of four spaces.
- T Prints the current time and date.

There may be any number of occurances of options within the tag line.

In generating the tag lines, RUNOFF uses the margins that are in effect when the Footing or Heading Command was issued. The tag line is not affected by subsequent changes to the margins.

4.2 Index Table: The Index Table is built via use of the INDEX command. It stores, in sorted order, phrases and the pages that they occur on. The Index Table can be printed with the PRINT INDEX command and permanently stored using the SAVE INDEX command.

4.3 TOC Table: The TOC (for Table of Contents) Table is built via the use of CHAPTER and SECTION commands. It stores the Sections Counter, Title and Page Number. This can then be printed with the CONTENTS command and permanently stored using the SAVE CONTENTS command.

4.4 Current Line Counter: The Current Line Counter keeps track of the number of lines printed on the page and is incremented every time a line is printed.

4.5 Current Page Counter: The Current Page Counter keeps track of the number of pages printed and is incremented every time a Page Break occurs.

4.6 Line Spacing Counter: The Line Spacing Counter keeps track of the number of blank lines between printed lines of text.

4.7 Sections Counter: The Sections Counter keeps track of the current section number for use by the CHAPTER and SECTION commands. The Sections Counter is multi-level; level 1 is commonly known as the chapter number. Subsequent levels enumerate subsections within the chapter. Used in the SECTION command, the chapter number, level 1, is displayed left of the period. Subsequent levels are displayed with periods separating them as diagrammed:

level1.level2.level3.level4.level5

When the counter for a specified level is incremented, all subsequent levels are set to zero.

4.8 Indent: Indent is set by the Indent Margin command and generates a number of spaces to preceed the line when printed. If Indent is negative, it will subtract spaces from the calculation of the left margin as defined by Left Margin, Indent, Temporary Indent, Offset Indent and Paragraph Indent.

4.9 Left Margin: The Left Margin is a counter like the other indents which generates a number of spaces to preceed the line when it is printed. Left Margin cannot be negative.

4.10 Paragraph Indent: The Paragraph Indent is set by the Paragraph Command and generates a number of spaces to preceed the line when printed. It is only used in the calculation of left margin on the first line of a paragraph, which is the first line after a Paragraph Break. If Paragraph Indent is negative, it will subtract spaces from the calculation.

4.11 Temporary Indent: Temporary Indent is set by the Indent command and generates a number of spaces to preceed the line when printed. If Indent is negative, it will subtract spaces from the calculation of the left margin as defined by Left Margin, Indent, Temporary Indent, Offset Indent and Paragraph Indent.

Temporary Indent is effective for one line only, and is reset to zero after it is used.

4.12 Text Breaks: Text breaks are caused by RUNOFF commands which change the mode of formatting. This requires that those words which are waiting to be output as part of an unfinished line to be output. Then the mode is changed as specified by the RUNOFF command, then formatting and output continues.

4.13 Paragraph Breaks: The beginning of a paragraph can be indicated in the source text by either an empty line or a line starting with a blank. In addition, RUNOFF commands can cause, or specify, the end of a paragraph. A Paragraph Break consists of a Text Break, followed by the number of blank lines indicated by the Line Spacing Counter. The left margin is offset by the number of spaces indicated by the Paragraph Indent. If the Paragraph Indent is negative, then the calculation of the left margin is decremented by the the absolute value of Paragraph Indent. In addition, Line Length is temporarily decremented by the value of Paragraph Indent.

4.14 Page Breaks: A Page Break can be caused by RUNOFF commands or when the Current Line Counter plus the number of lines to be printed in the Footing Tag exceeds the Page Depth. A Text Break is generated, if needed, and then the Footing Tag is printed. The page is ejected and the Heading Tag is printed. The Current Page Counter is incremented. The Current Line Counter is incremented to the number of lines printed in the Heading Tag.

4.15 Boldface Mode: When in Boldface mode, every character is overprinted again to emphasis it. The number of overstrikes can be varied from the default, one, by specifying the number of overstrikes as an option in the RUNOFF TCL statement.

4.16 Box Mode: When in Box Mode, output text is bracketed on the left and right with vertical bars (|). The left and right columns of the box are defined by the ".BOX" command that turned the Box Mode command on.

4.17 Capitalize Sentence Mode: When in Capitalize Sentence Mode, the first character of each word following a period or a question mark is capitalized.

4.18 Fill Mode: When in Fill mode, words are taken from the source item and placed in a buffer which represents one line of output. When RUNOFF attempts to put a word in this buffer that would cause the buffer to be longer then the line width, the buffer will be output and that word will be placed at the beginning of the next line buffer.

4.19 Highlight Mode: When in Highlight mode, a character is printed two columns to the right of the right margin. The character printed is specified as an argument in the ".HILITE"

command.

4.20 Justify Mode: When in Justify mode, each line before it is output, will be adjusted so that the end of the last word on the line will end directly at the right margin, or width of the line. Blanks are inserted randomly between words to accomplish this. The line immediately preceeding a text break is not justified.

Justify mode "on" implies that Fill mode is also "on".

4.21 Lower Case Mode: When in Lower Case Mode, RUNOFF will output all characters in lower case, except as directed by Capitalize Sentence Mode.

4.22 Underline Mode: When in Underline mode, an underline character, "_" is printed below every character. If Fill Mode is on, blanks are NOT underlined.

4.23 Upper Case Mode: When in Upper Case Mode, RUNOFF will output all characters in upper case, except as directed by Capitalize Sentence Mode.

4.24 Initial Conditions: When RUNOFF is started, the tags, counters and modes are set as followed:

Boldface Mode	off
Box Mode	off
Capitalize Sentence Mode	on
Current Line Counter	.0
Current Page Counter	.1
Fill Mode	on
Footing	null
Heading	null
Highlight Mode	off
Indent	0
Justify Mode	on
Left Margin	0
Line Length	70
Line Spacing Counter	.1
Lower Case Mode	off
Paper Length	*1
Paragraph Indent	5
Sections Counter	1.0
Tab Stops	not set
Temporary Indent	0
Underline Mode	off
Upper Case Mode	off

*1: Initialized to the Page Depth set by the TERM TCL command.

5.0 RUNOFF Command Definitions

5.1 Comment: The Comment command causes the RUNOFF processor to ignore any text following the "*". This can be used by the user to insert comments about special conditions. The format of this command is:

```
.*{text}
```

5.2 Begin Page: The Begin Page command causes a Page Break. Permissable formats of the command are:

```
.BP
.BEGIN PAGE
```

5.3 Box: The Box command causes a Text Break and turns the Box Mode on or off. To turn box mode on, you must specify:

```
.BOX leftedge,rightedge
```

The command to turn Box Mode off is:

```
.BOX OFF
```

Turning the Box mode on or off also prints a line of hyphens (-) between the "leftedge" and "rightedge".

5.4 Break: This command causes a text break. Permissable formats of the command are:

```
.B
.BREAK
```

5.5 Capitalize Sentence: The Capitalize Sentence command turns on Capitalize Sentence Mode. Permissable formats of the command are:

```
.CS
.CAPITALIZE SENTENCES
```

5.6 Center: The Center command causes a text break and then prints text centered between the left and right margins. Permissable formats of the command are:

```
.C
.CENTER
```

Only the next line is centered.

5.7 Chain: The Chain command causes the RUNOFF processor to use another, specified, source item for text. The format of the Chain command is:

.CHAIN {{DICT }filename} itemname

If "filename" is not specified, then the filename is assumed to be the same file as the current source item. Control is never returned to the current source item.

5.8 Chapter: The Chapter command causes a page break; prints the literal "CHAPTER" followed by the current chapter number (Sections Counter, level 1); increments the chapter number; skips a line; prints the optional chapter title; skips a line. The format of the command is:

.CHAPTER {chapter#}{title}

If "chapter#" is specified, then the current chapter number is reset to "chapter#". An entry is made in the TOC Table.

5.9 Contents: The Contents Command causes a Page Break; prints the literal:

Table of Contents

centered between left and right margins; skips two lines; prints the TOC Table based on previous CHAPTER and SECTION commands. The format of the command is:

.CONTENTS

5.10 Crt: The CRT command redirects subsequent output to the terminal. The format is:

.CRT

5.11 End Case: The End Case command turns the Upper Case and Lower Case Modes both to "off". Permissable formats of the command are:

.EC
.END CASE

5.12 Fill: The Fill command sets the Fill Mode on. Permissable formats of the command are:

```
.F
.FILL
```

If the Fill Mode is currently off, a Text Break is generated.

5.13 Footing: The Footing command specifies text and information to be displayed at the bottom of each page. The format is:

```
.FOOTING
```

The next line of text following the Footing command is used as the Footing Tag. Heading and Footing Tags can have data automatically inserted (such as page number) if the appropriate options are set. See the section on Heading and Footing Tags.

The Footing Tag takes effect at the next Page Break to occur, except for the special case of when nothing has been printed yet. In that case, the Footing Tag takes effect after the first Page Break.

The Footing Tag is reset to null if the line following the Footing command is a RUNOFF command line.

5.14 Heading: The Heading command specifies text and information to be displayed at the top of each page. The format is:

```
.HEADING
```

The next line of text following the Heading command is used as the Heading Tag. Heading and Footing Tags can have data automatically inserted (such as page number) if the appropriate options are set. See the section on Heading and Footing Tags.

The Heading Tag takes effect at the next Page Break to occur.

The Heading Tag is reset to null if the line following the Heading command is another RUNOFF command line.

5.15 Hilite: The Hilite command turns the Highlight Mode on and off. The format of the command to turn the mode on is:

```
.HILITE character
```

The "character" specified will be printed in the right margin. To turn the mode off the command is:

```
.HILITE OFF
```

5.16 Indent: The Indent Command causes a Text Break and adjusts the indent from the left margin for the next line only. The format is:

```
.I spaces
.INDENT spaces
```

The "spaces" can be either a positive or negative number and may cause the indent to go negative, which would cause the output of text to start before the left margin by the absolute value of indent.

5.17 Indent Margin: The Indent Margin Command causes a Text Break and adjusts the indent from the left margin. Permissable formats of the command are:

```
.IM spaces
.INDENT MARGIN spaces
```

The "spaces" can be either a positive or negative number and may cause the indent to go negative, which would cause the output of text to start before the left margin by the absolute value of indent. The specification "spaces" is cumulative in that it is added to the current value of indent.

5.18 Index: The Index command places the following term with the current page number into an index table which can be printed out later with the "Print Index" command. The format is:

```
.INDEX term { term...}
```

If the term contains blanks, it must be enclosed in double quote marks.

5.19 Input: The Input command prompts for input from the terminal and uses the response as text to be processed. The format is:

.INPUT

5.20 Justify: The Justify command turns the Justify and Fill Modes on. Permissable formats of the command are:

.J
.JUSTIFY

If the Fill Mode is currently off, a Text Break is generated.

5.21 Left Margin: The Left Margin command causes a Text Break and sets the left margin column. Permissable formats of the command are:

.LM column
.LEFT MARGIN column

The "column" specification must be a positive whole number.

5.22 Line Length: The Line Length command causes a Text Break and sets the width of the output line. The format is:

.LINE LENGTH columns

The "columns" specification must be a positive whole number.

5.23 Lower Case: The Lower Case command turns the Upper Case Mode off and the Lower Case Mode on. Permissable formats of the command are:

.LC
.LOWER CASE

5.24 Line Printer: The Line Printer command directs subsequent processing to output to the system spooler. The format is:

.LPTR

5.25 No Capitalize Sentence: The No Capitalize Sentence Command turns the Capitalize Sentence Mode off. Permissable formats of the command are:

```
.NCS
.NOCAPITALIZE SENTENCES
```

5.26 Nofill: The Nofill command turns the Fill and Justify Modes off. Permissable formats of the command are:

```
.NF
.NOFILL
```

If the Fill Mode is currently on, a Text Break will be generated.

5.27 Nojustify: The Nojustify command turns the Justify Mode off. Permissable formats of the command are:

```
.NJ
.NOJUSTIFY
```

If the Justify Mode is currently on, a Text Break will be generated.

5.28 Noparagraph: The No Paragraph command disables the processing of Paragraph breaks. The format is:

```
.NOPARAGRAPH
```

5.29 Page Number: The Page Number command sets the current page number to the specified number. The format is:

```
.PAGE NUMBER page
```

The "page" specification must be a positive whole number.

5.30 Paper Length: The Paper Length Command sets the number of lines on a page to the specified number. The format is:

```
.PAPER LENGTH lines
```

The "lines" specification must be a positive whole number.

5.31 Paragraph: The Paragraph Command specifies the number of spaces that the first line of a paragraph is indented. Permissable formats are:

.P spaces
.PARAGRAPH spaces

The "spaces" specification can be either a positive or negative whole number. A negative number indicates the first line of a paragraph begins the absolute value of "spaces" columns before the left margin.

5.32 Pfile: The PFILE command directs further processing to be output to the specified spooler print file. The format is:

.PFILE file#

The "file#" specification must be a whole number between 0 and 125.

5.33 Print: The Print command prints the following source line on terminal. The format is:

.PRINT

5.34 Print Index: The Print Index command causes the index table built via previous Index commands to be printed. The format is:

.PRINT INDEX

5.35 Read: The Read command causes the RUNOFF processor to use another, specified, source item for text. The format of the command is:

.READ {{DICT }filename} itemname

If "filename" is not specified, then the filename is assumed to be the same file as the current source item. When the RUNOFF is finished processing the "read" source item, it continues processing the current source item.

5.36 Readnext: The Readnext command extracts the next element from the implicit itemlist and uses it as source text. The format is:

.READNEXT

5.37 Save Index: The Save Index command stores the index table built via previous Index commands as an item in a specified file. The format is:

.SAVE INDEX filename

The item id will be the same as the source item id.

5.38 Section: The Section command causes a Text break then the Sections Counter is displayed and incremented, followed by the optional text. The Sections Counter, Text and Page Number are stored in the TOC Table. A Paragraph Break is then generated. The format is:

.SECTION level {title}

5.39 Set Tabs: The Set Tabs command defines tabulation columns which are used with the "<" and ">" embedded subcommands. The format is:

.SET TABS column{,column...}

5.40 Skip: The Skip command causes a text break and prints a specified number of blank lines. Permissable formats of the command are:

.SK lines
.SKIP lines

The number of blank lines printed is the specification "lines" multiplied by the Line Spacing Counter.

5.41 Space: The Space command causes a text break and prints a specified number of blank lines. Permissable formats of the command are:

.SP lines
.SPACE lines

The specification "lines" defines the number of blank lines generated, independent of the Line Spacing Counter.

5.42 Spacing: The Spacing command defines the vertical line spacing and is stored as the Line Spacing Counter. The format is:

.SPACING lines

5.43 Standard: The Standard command initializes various modes to the default values. The format is:

```
.STANDARD
```

The command performs the same actions as the following list of commands:

```
.CAPITALIZE SENTENCES
.FILL
.JUSTIFY
.FOOTING
.HEADING
.LEFT MARGIN 0
.LINE LENGTH 70
.PARAGRAPH 5
.END CASE
```

5.44 Test Page: The Test Page command verifies that the specified number of lines can be printed on the page. If not, a Page Break is generated. Permissable formats of the command are:

```
.TP lines
.TEST PAGE lines
```

5.45 Upper Case: The Upper Case command turns the Upper Case Mode on and the Lower Case Mode off. Permissable formats of the command are:

```
.UC
.UPPER CASE
```

6.0 Embedded Subcommands

RUNOFF provides for a set of special characters, or subcommands, which can be embedded within the source text to control modes.

6.1 Boldface: The Boldface subcommand turns the Boldface Mode as specified:

@ Next character only is printed in Boldface Mode.

@^ Boldface Mode is turned on.

@\ Boldface Mode is turned off.

6.2 Underline: The Underline subcommand controls the Underline Mode as specified:

& Next character only is printed in Underline Mode.

&^ Underline Mode is turned on.

&\ Underline Mode is turned off.

6.3 Upper Case: The Upper Case subcommand controls the Upper Case Mode as specified:

^ Next character only is printed in Upper Case Mode.

^^ Upper Case Mode is turned on.

6.4 Lower Case: The Lower Case subcommand controls the Lower Case Mode as specified:

\ Next character only is printed in Lower Case Mode.

\\ Lower Case Mode is turned on.

6.5 Literal Lead-in: The Literal lead-in subcommand specifies the next character to be treated as a literal and not to be considered as an embedded subcommand. The Literal lead-in subcommand is a "_", underline character.

If the literal character is a space and Justify Mode is on then RUNOFF will not insert extra blanks at this point to create a justified line.

6.6 Left Tab: The Left Tab Subcommand causes spaces to be inserted into the line so that the next word will begin in the next specified tabulation column. If there is no corresponding tabulation column specified, a single space will be inserted.

6.7 Right Tab: The Right Tab Subcommand causes the next word to be right justified so that it ends in the next specified tabulation column. If there is no corresponding tabulation column specified, a single space will be inserted.

This is the last page.