SMA STANDARD

SMA/PROC Language Specification

SMA: 501

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 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.

4.10 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.

4.20 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