WORDS WERE JUST DATA UNTIL...

S T R I N G
M A S T E R

99/4A Program Environment
100% Assembly Language LINKs for Extended BASIC

"...one of the most valuable valuable software development tools the 99/4A has seen since the Extended BASIC cartridge."
- *Computer Shopper*

Another Fine Product From

BYTEMASTER COMPUTER SERVICES
171 MUSTANG STREET
SULPHUR, LA 70663-6724

SYSTEM REQUIREMENTS:

TI-99/4A or Compatible
TI Extended BASIC or Compatible
Memory Expansion
Disk Drive

ONLY $19.95

STRING MASTER

TABLE OF CONTENTS

See the program DEMO on the STRING MASTER diskette for additional ideas on using the functions of STRING MASTER.

REGISTRATION FORM

REGISTRATION #SM0023

Name_____

Street_____

City_____

State_____

Zip_____

Country_____

Vendor_____

Date of Purchase_____

Mail to:

Bytemaster Computer Services
171 Mustang Street
Sulphur, LA 70663-6724

STRING MASTER Overview

STRING MASTER is a set of Assembly Language routines accessible from TI Extended BASIC (or derivations therefrom, such as Triton Super Extended BASIC). String Master routines provide for handling macro equivalents of standard XB functions, thereby greatly reducing program development time and program run time. Furthermore, STRING MASTER's speed and practical design provide an environment that is conducive to creative programming.

For those of you unfamiliar with accessing Assembly routines from XB, it is really quite simple. For instance, it would be possible to write an Assembly equivalent of XB's CALL HCHAR. The regular HCHAR is used like this:

```
CALL HCHAR(1,1,32,768)
```

An Assembly equivalent could be accessed like this:

```
CALL LINK("HCHAR",1,1,32,768)
```

As you can see, there is very little difference, so don't be intimidated by the words "Assembly Language"! Simply follow the guidelines of this manual and you will find programming with STRING MASTER to be as easy as programming in XB.

STRING MASTER Conventions

All routines can be fully utilized with any standard 99/4A system with disk, memory expansion and Extended BASIC.

TI Extended BASIC and direct derivations therefrom are fully supported (MYARC XB is not supported because it is internally radically different).

Both OPTION BASE 0 and OPTION BASE 1 are fully supported.

Any string type (direct string, string variable, string array element or string array) is valid for any string parameter.

Any numeric type (direct number, numeric variable, numeric array element or numeric array) is valid for any numeric parameter.

Both string and numeric arrays can utilize any number of dimensions supported by XB, 1 to 7.

Complete error checking with standard XB error messages.

Numeric integers within a maximum range of -32768 to 32765, inclusive, are fully supported.

CALL LINK's utilize standard XB syntax. In other words, complex expressions may be used only for passing parameters to (not from) assembly routines and may not pass entire arrays (examples of valid complex expressions are "A$&B$" and "A+B").

Reminders

STRING MASTER provides many powerful features for handling strings and is designed to follow standard protocols to the extent possible within an enhancement package. Still, there are routine cautions to be observed.

The 99/4A has a limited area for storing strings, but system software recovers string memory no longer being utilized. Therefore, it is often very advantageous to use functions available in STRING MASTER to minimize the lengths of strings, so that memory will remain available for additional string processing. User programs should be thoroughly tested to ensure that system string memory space cannot be exceeded. When string memory space is exceeded, a "Memory Full in Line XXXXX", where "XXXXX" is a line number, error message will be issued as a standard system error.

User programs should be thoroughly tested before being used to over-write important files. This is a standard tenant of program development that is especially relevant in using environments previously unfamiliar to the programmer.

Conventions of the STRING MASTER Manual

Each STRING MASTER function is described in detail. Near the beginning of each description, the proper CALL LINK format for linking to the assembly routines is given. The format utilizes the following abbreviations for the data types of the LINK parameters:

SV = string variable, direct string or string array element
SA = string array
NV = numeric variable, direct number or numeric array element
NA = numeric array
DM = does not matter (any data type)

As previously described, string parameters can utilize any string type and numeric parameters can utilize any numeric type. The LINK formats are supplied to indicate the preferred data type. Where a non-array is indicated, usage of an array will pass the first element of the array to the function. Where an array is indicated, usage of a non-array will have the same effect as using an array dimensioned to one element. Functions with a format that specifies an option, indicated by "OR" (for example, "SV OR SA"), will act upon the two data types differently, with subsequent documentation indicating the exact manner each is handled. "DM" indicates that it does not matter what data type is utilized; the function will act or not act depending on whether the parameter is used, regardless of the data type. Below the LINK format is a line indicating whether each parameter is R)equired, O)ptional, or a "S)ub-parameter" (see below), followed by a line indicating paired optional parameters (denoted with "\/").

Following the LINK format is the parameter description list. The specific usage of each parameter, including any range limitations, is described in the parameter description list. The word "parameter" is abbreviated "PARM" in the list. Thus, PARM 1 refers to the first parameter in the LINK. Note that some parameters are listed as what might be casually referred to as "sub-parameters". The number of sub-parameters corresponds to the number of dimensions of a specified array. If a non-array is used where an array is preferred, then one sub-parameter must be used for values passed from the assembly routines and/or no sub-parameters should be used for values passed to the assembly routines. Sub-parameters are numbered in the parameter list as a parameter with a suffix (for example, "PARM 3A1 through 3A7").

The function description then continues with a list of any optional parameters, including any parameters that are dependent upon usage of other parameters.

Finally, the function description can include Comments, Usage, Applications, Cautions and Examples to clarify the exact way to use the function.

Loading STRING MASTER

METHOD 1

STRING MASTER can be loaded with the standard Extended BASIC CALL's for loading Assembly routines as relocatable code to Low Memory, as follows:

```
100 CALL INIT
110 CALL LOAD("SM/O")
```

METHOD 2

While the standard CALL LOAD method provides full integrity of the XB environment, the XB loader is very slow. To facilitate a faster load, Todd Kaplan's popular "XBAL" load method has been employed for those users who prefer simplified, fast loading. "XBAL" loads (to High Memory) "hidden" assembly code along with an XB program and the XB program triggers moving the assembly code to Low Memory (beginning at >24F4, the standard address for loading immediately following CALL INIT). While a programmer can add code to an XBAL program, it is more common to RUN or OLD another program, thereby freeing the area containing redundant assembly code. After an XBAL program completes, the XB environment is standard, as if the standard CALL LOAD process had been employed. The STRING MASTER XBAL program is the "LOAD" file on the STRING MASTER disk. Thus, inserting the STRING MASTER disk in drive 1 and selecting Extended BASIC from the main system menu will load the STRING MASTER assembly routines. Optionally, use (drive 1 is merely an example, the drive number can be any available drive):

```
RUN "DSK1.LOAD"
```

METHOD 3

The third loading method is somewhat less standard and is therefore recommended for somewhat advanced users. A method similar to that described by George Steffen in *GENIAL TRAVeIER* diskazine has been used to allow having the STRING MASTER routines reside in High Memory. To accomplish the High Memory residency, the system pointers to the XB program are manipulated. Once that is done, a few programs that use CALL LOAD's to poke values to memory may not function as intended. Also, OLD or RUN will reset the pointers. To load the High Memory routines, use:

```
OLD "DSK1.LOADH"
```

To write XB code for the High Memory environment, OLD in LOADH, key in the program normally and SAVE the program. To use or modify a program in the High Memory environment that was not written for the High Memory implementation of STRING MASTER, OLD in the program (OLD "DSK1.PROG"), SAVE the program in MERGE format (SAVE "DSK1.MYPROGRAM",MERGE), OLD in LOADH (OLD "DSK1.LOADH"), MERGE in the program (MERGE "DSK1.MYPROGRAM"), then SAVE normally (SAVE "DSK1.PROGRAM"), thereafter loading the program normally. With STRING MASTER residing in High Memory, Low Memory is available for assembly code (the standard XB LOAD must be used to avoid destruction of the STRING MASTER REF/DEF table).

APENDA

APpEND Array

```
CALL LINK("APENDA",SA,SA,NV,NV,NV,NV,NV,NV,NV,NV,NV,SV,SV,SV,NV,DM)
                  R  R  S  S  S  S  S  S  S  O  O  O  O  O  O  O
                                                \/    \/
```

PARM 1: Source; array to append to another array.
PARM 2: Destination; array at which to append.
PARM 3A1 through 3A7: From 1 to 7 parameters, one for each dimension of the destination array, for indicating the element at which to begin an append. Use no sub-parameters for non-array destinations.
PARM 4: SEG$ position (range: 1 to 255)
PARM 5: SEG$ length (range: 1 to 255)
PARM 6: Operator for selectively engaging the SEG$ function, can be one of the following (must be one character):

- = (equal)
- > (greater than)
- < (less than)
- G (greater than or equal)
- L (less than or equal)
- X (not equal)
- A (all) (any character not otherwise used can be substituted for "A")

PARM 7: Match String, for matching the SEG$ through an operator
PARM 8: SEG$ Qualifier; determines whether the results of the Operator function will be used to utilize the entire string or the SEG$. Use of "S" qualifies the SEG$, any other string qualifies the entire string.
PARM 9: Count of the strings passing the Operator test.
PARM 10: Count Only; used to obtain only a count, with no append. Omit PARM 10 to actually append.

Optional Parameters: 4 through 10. Parameters must be in sequence, with the only omissions permitted being sub-parameters that are not applicable and options at the end of the parameter list that are not used. PARM 4 is dependent on PARM 5. PARM 6 is dependent on PARM 7.

Comments: APENDA is one of the most powerful functions of String Master and is also the most complex in structure. Optional parameters allow learning the capabilities of APENDA a step at a time, as well as permitting a full range of capabilities.

Usage: Copies strings from one array to another array or from an array into the same array. Allows a function equivalent to XB's SEG$ at the time of the copy. The string result of the SEG$ can optionally be tested against a Match String for copying strings selectively. A count of the elements copied can be obtained. When selectively copying, either the original string or the SEG$ can be copied. Using the Count Only feature, the append can be by-passed and yet a count of the elements that would otherwise have been copied can be obtained. The append can begin at any element of the destination array. The append automatically stops when either the entire source array has been copied or the end of the destination array is reached (i.e., "string stack overflow" errors will not be encountered). Note that copying is done sequentially into the destination array, so that there is no implied correlation between corresponding element numbers of the two arrays.

Applications: Useful for database operations in which a field is derived as a SEG$ of a string. Can also be used to obtain counts of strings matching certain criteria. Many other applications possible, including clever manipulations of character patterns.

Cautions: Appending an array into itself while not beginning at the first element or while copying matched strings can yield unanticipated results. The function may repeatedly pick up a particular element or otherwise perform in a manner other than what might be expected. Appending an array into itself is recommended only for purposes of obtaining a Count Only or SEG$.

Example 1: The following is equivalent to A$=B$.

```
100 CALL LINK("APENDA",B$,A$)
```

Example 2: This example will provide the equivalent of A$(I,J)=SEG$(A$(I,J),1,5) where I and J are FOR-NEXT loop counters that cycle through the entire array, for all elements of the array.

```
100 CALL LINK("APENDA",A$(,),A$(,),0,0,1,5)
```

Example 3: The following will copy all occurances of "AAAAA" as the first five letters of an element to B$().

```
100 CALL LINK("APENDA",A$(),B$(),0,1,5,"=","AAAAA")
```

Example 4: This code will copy all elements of A$() that begin with "AAAAA" to B$().

```
100 CALL LINK("APENDA",A$(),B$(),0,1,5,"=","AAAAA","")
```

Example 5: This one will do everything in Example 4, plus it will provide a count of the elements copied.

```
100 CALL LINK("APENDA",A$(),B$(),0,1,5,"=","AAAAA","",COUNT)
```

Example 6: Finally, this example will provide the same count as in Example 5, but will not append.

```
100 CALL LINK("APENDA",A$(),B$(),0,1,5,"=","AAAAA","",COUNT,C_ONLY)
```

BEEP

BEEP

```
CALL LINK("BEEP")
```

Usage: Used to generate a BEEP tone.

Applications: Useful for providing emphasis with a positive connotation.

BINHEX

BINary to (ASCII) HEXadecimal conversion

```
CALL LINK("BINHEX",SA,SA)
                   R  R
```

Parm 1: Binary string (a string consisting of characters in the range 0 to 255, inclusive).
Parm 2: Hex string (a string of characters from the set "0123456789ABCDEF")

Comments: See also HEXBIN.

Usage: Used to convert a binary string to a hexadecimal string. Each binary digit is converted to two hexadecimal digits. For instance, character 255 would be a single character in the binary string, but would become "FF" in the hexadecimal string.

Applications: Useful in conjunction with SMPEEKV to obtain character patterns SMPEEKV and BINHEX can be used together for the equivalent of XB's CHARPAT, but the STRING MASTER functions may utilize an array, while CHARPAT is limited to use with strings only. Other memory peeking applications are possible.

Cautions: The binary string should be no longer than 127 characters. Because BINHEX yields a twice-as-long string in the destination (the hex string), use of a binary string greater than 127 characters will cause a "String Truncated" error and termination of the program.

Example:

```
100 A$(0)=CHR$(25)&CHR$(48)
110 CALL LINK("BINHEX",A$(),B$())
```

CIRCUL

CIRCULar rotation of an array

```
CALL LINK("CIRCUL",SA,NV)
                  R  R
```

Parm 1: Array to rotate.
Parm 2: Number of single-element rotations to accomplish.

Usage: Used to move the first element of an array to the last element position and move all other elements down one position.

Applications: Useful for changing the screen display position of an array element while using WINDOW. For instance, if an array has 48 elements, then 24 rotations will swap a "hidden" 24-row window with the one being displayed on the next WINDOW of that array.

Cautions: While CIRCUL can "hide" elements, it will not eliminate those elements, so that functions such as SORTA may bring an element back into a more predominant role than casually anticipated.

Example:

```
100 DIM A$(47) :: CALL LINK("CIRCUL",A$(),24)
```

CONCTL

CONCaTenate Left

```
CALL LINK("CONCTL",SA OR SV,SA)
                   R     R
```

Parm 1: Source array
Parm 2: Destination array

Comments: See also CONCTR, APENDA, MAXLEN

Usage: Used to concatenate (equivalent to using the "&" in XB) the elements of

one array to the corresponding elements of another array. Or, if a direct string, string variable or string array element is used as the source, the string will be concatenated to every element of the array. The source string is placed on the left side of the destination string.

Applications: Useful for database operations to rebuild a string taken apart with APENDA or XB's SEG$. For instance, if A$() is an array of first names and B$() is an array of last names, using CONCTL will place both the first and last name in B$() if B$() is the destination.

Cautions: Exceeding the 255-byte limit of a single string will yield a "String Truncated" error message and terminate program execution in an error state. If all strings are 127 characters or less, no error can occur (MAXLEN can be used to test for maximum length within an array). Careful program planning is required in using CONCTL.

Example 1, places an "A" preceding the first character of each element of A$():

```
100 CALL LINK("CONCTL","A",A$())
```

Example 2, concatenates each element of A$() to the left side of the corresponding element of B$():

```
100 CALL LINK("CONCTL",A$(),B$())
```

CONCTR

CONCaTenate Right

```
CALL LINK("CONCTR",SA OR SV,SA)
                    R     R
```

Parm 1: Source array.
Parm 2: Destination array.

Comments: See also CONCTL.

Usage: Used to concatenate an array, element for element, with another array. The source will be placed to the right side of the destination and the concatenated string will reside in the destination. Use of a direct string, string variable or string array element as the source will place the source string to the right of every element in the destination.

Applications: Useful for database operations.

Cautions: Exceeding the 255-byte limit of a single string will result in an error message and termination of the program in an error state.

Example 1, places an "A" following the last character of each element of A$():

```
100 CALL LINK("CONCTR","A",A$())
```

Example 2, concatenates each element of A$() to the right side of the corresponding element of B$():

```
100 CALL LINK("CONCTR",A$(),B$())
```

EOANN

End-Of-Array, Non-Null

```
CALL LINK("EOANN",SA,NV,NV,NV,NV,NV,NV,NV)
                  R  S  S  S  S  S  S  S
```

Parm 1: Array
Parm 2 A1 through A7: From 1 to 7 parameters, one for each dimension of the array, for indicating the last element that is not a null string.

Comments: See also APENDA

Usage: Used to find the last element in an array that is not a null string (a null string is a string with a length of zero).

Applications: Useful for reducing output by stopping at the last element that has what would typically be considered usable contents.

Cautions: EOANN finds the last non-null string. A null string can precede the non-null string within the array. Use APENDA with Operator "X" and Match String "" to eliminate nulls that precede the EOANN.

Example:

```
100 DIM A$(2,3,2)
110 A$(1,2,1)="HELLO"
120 CALL LINK("EOANN",A$(,,),A,B,C)
```

FIXLEN

FIX the LENgth of a string

```
CALL LINK("FIXLEN",SA,NV)
                   R  R
```

Parm 1: String (array).
Parm 2: Length, 0 to 255, for the string(s) (in the array).

Comments: See also MAXLEN.

Usage: Used to set all strings in an array to a specific length.

Applications: Useful for database applications where an array with strings of variable length are to be set to a like length for usage as a field. Also extremely useful for saving memory by setting an entire array to null strings (length of 0).

Cautions: Can truncate strings, so usage of MAXLEN may be appropriate.

Example:

```
100 A$(3)="HAMBURGER"
110 CALL LINK("FIXLEN",A$(),3)
```

FRAME

window FRAME

```
CALL LINK("FRAME",NV,NV,NV,NV,NV)
                  R  R  R  R  R
```

Parm 1: Beginning screen row. Range: 1 to 23.
Parm 2: Beginning screen column. Range: 1 to 31.
Parm 3: Ending screen row, must be larger than Parm 1. Range: 2 to 24.
Parm 4: Ending screen column, must be larger than Parm 1. Range: 2 to 32.
Parm 5: Character for frame. Range: 32 to 143.

Comments: See also WINDOW

Usage: Used to draw a hollow box.

Applications: Useful for framing a window.

Cautions: To avoid over-writing a window with a frame or vice-versa, remember to make the frame one character position larger than the window on each side.

Example:

```
100 CALL LINK("FRAME",1,3,3,30,143)
```

HEXBIN

(ASCII) HEXadecimal to BINary string conversion

```
CALL LINK("HEXBIN",SA,SA)
                  R  R
```

Parm 1: Hex string (a string of characters from the set "012345689ABCDEF")
Parm 2: Binary string. Range: characters 0 through 255, inclusive.

Comments: See also BINHEX.

Usage: Used to convert a binary string to a hexadecimal string. Each two hex digits are converted to a single binary digit. For example, "FF" would convert to character 255.

Applications: Useful for use with SMPOKEV to provide the equivalent of XB's CHAR, but STRING MASTER's functions can utilize an entire array, while CHAR is limited to a single string. Other memory poking applications are possible.

Example:

```
100 CALL LINK("HEXBIN","FF",A$)
```

HEXDEC

(ASCII) HEXadecimal to DECimal conversion
or
decimal to (ASCII) hexadecimal conversion

```
CALL LINK("HEXDEC",SA,NA) or CALL LINK("HEXDEC",NA,SA)
                  R  R                          R  R
```

Parm 1: If a string, Parm 2 must be numeric and vice-versa.
Parm 2: If numeric, Parm 1 must be a string and vice-versa.

Usage: Can be used to convert hex to decimal or decimal to hex. The first parameter is the source (to be converted) and the second parameter is the destination. The routine automatically determines the type of the first parameter and expects an appropriate type as the second parameter.

Applications: Can be used for converting memory addresses from base 16 to base 10 and vice-versa.

Cautions: The range limit is 4 digits for the hex string. Decimal values must fall within the range -32768 to 32767. Null strings convert to 0, thereby reducing program error terminations, though there is no decimal equivalent.

Example 1, Hex to decimal:

```
100 CALL LINK("HEXDEC","FFFF",A)
```

Example 2, decimal to Hex:

```
100 CALL LINK("HEXDEC",-32767,A$)
```

HONK

HONK

```
CALL LINK("HONK")
```

Usage: Generates a HONK tone.

Applications: Useful for providing emphasis with a negative connotation.

LTRIM

Left-side TRIM of unwanted characters

```
CALL LINK("LTRIM",SA,SV)
                  R  R
```

Parm 1: Array to be trimmed
Parm 2: Character(s) to trim, as a string.

Comments: See also TRIM.

Usage: Used to eliminate unwanted characters from the left side of a string. For instance, if Parm 2 is "E4", then all "E"'s and "4"'s at the beginning of each element of the array in Parm 1 will be removed up to the point where a character is not an "E" or a "4".

Applications: Useful for eliminating leading blanks and other characters that might not be desired for a particular situation.

Cautions: LTRIM does not eliminate all occurances of the specified characters, only the occurences at the beginning of the string. LTRIM'ing more than one character at a time should be done with caution, as one of the characters to be trimmed might appear to be safely imbedded within a string until other characters preceding it are eliminated.

Example:

```
100 A$(7)="  BEAR" :: CALL LINK("LTRIM",A$()," B")
```

MAXLEN

MAXimum LENgth present in an array

```
CALL LINK("MAXLEN",SA,NV)
                  R  R
```

Parm 1: Array.
Parm 2: Numeric return variable.

Comments: See also MINLEN, FIXLEN.

Usage: Used to determine the longest string in an array. Array equivalent XB's LEN.

Applications: Useful in determining the maximum required size for a database field.

Example:

```
100 A$(4)="TIGERS"
110 CALL LINK("MAXLEN",A$(),A)
```

MINLEN

MINimum LENgth of a non-null strings present in an array

```
CALL LINK("MINLEN",SA,NV)
                  R  R
```

Parm 1: String (array).
Parm 2: Numeric return variable.

Comments: See also MAXLEN, FIXLEN

Usage: Used to determine the shortest non-null string in an array.

Applications: Can be used for statistical analysis of the lengths of an array.

Example:

```
100 A$(5)="BARTER"
110 CALL LINK("MINLEN",A$(),A)
```

REBEEP

REBEEP

```
CALL LINK("REBEEP")
```

Usage: Sounds a beep tone repeatedly until a key is pressed.

REPLAC

REPLACe a segment of a string with a string

```
CALL LINK("REPLAC",SV,SA,SV)
                  R  R  R
```

Parm 1: String to replace.
Parm 2: String (array) in which to replace.
Parm 3: Replacement string.

Usage: Used to search through an array and replace every occurence of the string in Parm 1 with the string in Parm 3.

Applications: Excellent for replacing fields of a database or for correcting misspellings.

Cautions: Will not wrap from one element to the next, so that it cannot pick up on hyphenated words. If the replacement string is no longer than the string to replace, then the REPLAC has the potential for creating a string that exceeds XB's 255 character limit, so that a "String Truncated" error can result and cause program termination.

Example, replaces all occurences in A$() of "THEIR" with "THERE":

```
100 B$="THEIR" :: C$="THERE"
110 CALL LINK("REPLAC",B$,A$(),C$)
```

SEARCH

SEARCH an array for a string

```
CALL LINK("SEARCH",SV,SA,NV,NV,NV,NV,NV,NV,NV,NV,NV,NV,NV,NV,NV,NV)
                  R  R  S  S  S  S  S  S  S  S  S  S  S  S  S  S
```

Parm 1: String for which to search.
Parm 2: Array in which to search.
Parm 3 A1 through A7: From 1 to 7 parameters (no PARM 3 sub-parameters if PARM 2 is not an array), one for each dimension of the array in which to search, for indicating the element at which to begin the search.
Parm 4 A1 through A7: From 1 to 7 parameters, one for each dimension of the array in which to search, for indicating the element at which a match is found. If no match is found, those parameters required will return -1 (if PARM 2 is not an array, one PARM 4 sub-parameter is still required for returning 0 or -1).

Usage: Used to SEARCH an array for a string. Because allowances are made for both the begin and find elements, searches can continue where a previous search left off.

Application: Excellent for finding specific text passages or for searching database.

Cautions: Does not indicate multiple occurences of a string within an element.

Example 1, returns 4 in variable A, indicating that "COMPUTE" is found within the string "COMPUTER" at element 4 of A$(). The SEARCH begins at element 0 of A$().

```
100 A$(4)="COMPUTER"
110 CALL LINK("SEARCH","COMPUTE",A$(),0,A)
```

SELECT

view and SELECT an element in an array

```
CALL LINK("SELECT",SA,NV,NV,NV,NV,NV,NV,NV,NV)
                  R  R  S  S  S  S  S  S  S
```

Parm 1: Array in which to SELECT.
Parm 2: Beginning screen row for display, 1 to 14 (11 rows dislayed).
Parm 3 A1 through A7: From 1 to 7 parameters, one for each dimension of the array in which to SELECT, for indicating the element at which a selection is made.

Usage: Can be used with the keyboard or either joystick (1 or 2). Uses the up and down arrow keys or up and down on the joysticks to change the element displayed in its own screen window. Up decreases the element number, down increases the element number. When either end of the array is reached, that end of the array is displayed. The window can be placed beginning at row 1 through row 14. The first row displayed is the element number (the screen display number is valid only for single-dimension arrays, multi-dimension arrays display one number, as if a single-dimension array). The next 10 rows displayed are the string contents of that element. The longer up or down is depressed, the faster the sequencing through the elements (up to a maximum speed that is marginally readable). Keystrokes are initially handled rather slowly, providing ample time for reacting. When <ENTER> or the fire button of either joystick is depressed, the element selected is denoted in Parm 3 A1 through A7.

Applications: Useful for situations in which the contents of an array must be visually inspected, where the exact search parameters may initially be unknown. Can also be used for icons.

Cautions: When using multi-dimensional arrays, disregard the element number displayed on the screen, as it is the one-dimensional equivalent. When an actual selection is made, the proper multi-dimensional element will be properly denoted in Parm 3 A1 through A7.

Example, displays an element of A$() at row 1 and the contents of the element at rows 2 through 14; returns the number of the selected element in A

```
100 CALL LINK("SELECT",A$(),1,A)
```

SMPEEK

String Master PEEK of (CPU) memory

CALL LINK("SMPEEK",NV,NV,SA,NV)

Parm 1: CPU memory address to begin peek. Range: -32768 to 32767.
Parm 2: Bias to add to each byte peeked. Range: 0 to 255.
Parm 3: String array into which the peeked values are placed as a string.
Parm 4: Bytes per element. Range: 1 to 255.

Comments: See also SMPOKE.

Usage: Will peek as many bytes as desired, limited only by available string memory space. Will terminate when the number of bytes specified are peeked, when the specified array is filled or when string memory is full.

Applications: Can be used to examine ROM and RAM.

Cautions: SMPEEK will terminate with a "MEMORY FULL" error if string memory space is filled. Also, the string array used must be dimensioned to an adequate number of elements to hold the bytes to be peeked else the routine will terminate prior to peeking the specified number of bytes. As with CALL PEEK, some addresses can "lock up" the console!

Example 1, will peek 24 bytes from the beginning of High Memory, placing 8 bytes in each string array element of A$() with no bias.

```
100 CALL PEEK(-24576,0,24,A$(),8)
```

SMPOKE

String Master POKE of (CPU) memory

```
CALL LINK("SMPOKE",NV,NV,NV,SA)
                   R  R  R  R
```

Parm 1: Address at which to begin a poke. Range: -32768 to 32767.
Parm 2: Bias to add to each byte poked. Range: 0 to 255.
Parm 3: Number of bytes to poke.
Parm 4: String (array) to poke.

Usage: Will poke as many bytes as desired from an array. This is the string array equivalent of XB's function to poke values, LOAD.

Applications: Can be used to change RAM values.

Cautions: Writing to some addresses will lock up the console, requiring that a reset of the console be performed, normally accomplished by turning the console power off and back on.

Example, pokes the string "HELLO" at the beginning of High Memory, with no bias.

```
100 A$(0)="HELLO"
110 CALL LINK("SMPOKE",-24576,0,5,A$())
```

SORTA

SORT in Ascending order

```
CALL LINK("SORTA",SA)
                  R
```

Parm 1: String array to be sorted.

Comments: See also SORTAN, SORTD.

Usage: Sorts any string array into ASCII order.

Applications: Useful for database and other operations.

Cautions: SORTA is a very simple sort routine. Complex sorting is beyond the scope of this program, due to memory limitations. For complex, multi-level sorts, use a package such as Andy Deshoff's BasicSort 2.0.

Example, sorts A$() in ASCII sequence.

```
100 CALL LINK("SORTA",A$())
```

SORTAN

SORT in Ascending order, Nulls last

```
CALL LINK("SORTAN",SA)
                  R
```

Parm 1: String array to be sorted.

Comments: See also SORTA, SORTD.

Usage: Sorts any string array into ASCII order, except null strings are placed last.

Applications: Useful for database and other operations.

Cautions: SORTAN is a very simple sort routine. Complex sorting is beyond the scope of this program, due to memory limitations. For complex, multi-level sorts, use a package such as Andy Deshoff's BasicSort 2.0. Unlike standard sort routines, this one will place null strings last.

Example, sorts A$() in ASCII sequence with nulls last.

```
100 CALL LINK("SORTAN",A$())
```

SORTD

SORT in Descending order

```
CALL LINK("SORTD",SA)
                  R
```

Parm 1: String array to be sorted.

Comments: See also SORTA, SORTAN

Usage: Sorts any string array into descending ASCII order.

Applications: Useful for database and other operations.

Cautions: SORTD is a very simple sort routine. Complex sorting is beyond the scope of this program, due to memory limitations. For complex, multi-level sorts, use a package such as Andy Deshoff's BasicSort 2.0.

Example, sorts A$() in descending ASCII sequence.

```
100 CALL LINK("SORTD",A$())
```

STRINC

STRing equivalents of values INCremented

```
CALL LINK("STRINC",SA,NV,NV)
                   R  R  R
```

Parm 1: String (array) into which to place string equivalents of values.
Parm 2: Beginning numeric value. Range: -32768 to 32767.
Parm 3: Increment value from element to element. Range: -32768 to 32767.

Usage: An approximate string array equivalent of XB's STR$, but allowing incrementing the initial value through an array.

Applications: Can be used within a program that writes a program to assign line numbers.

Cautions: Function will terminate when 32767 is reached or exceeded or when the last available element of the string array is reached.

Example, will place "100" in A$(0), "110" in A$(1), "120" in A$(2).

```
100 DIM A$(2)
110 CALL LINK("STRINC",A$(),100,10)
```

SVPEEK

String (Master) Video (Display Processor) memory PEEK

```
CALL LINK("SVPEEK",NV,NV,NV,SA,NV)
                  R  R  R  R  R
```

Parm 1: VDP address at which to begin to peek. Range: 0 to 16383.
Parm 2: Bias, a value to be added to each byte peeked. Range: 0 to 255.
Parm 3: Number of bytes to be peeked.
Parm 4: The string (array) into which to place the peeked values, in string form.
Parm 5: Number of bytes per element. Range: 1 to 255.

Comments: See also SVPOKE.

Usage: Can be used to peek values from VDP.

Applications: SVPEEK can be used to peek the addresses of the Pattern Descriptor Table, then BINHEX can be used to convert the array into a format usable by XB functions such as CALL CHAR.

Cautions: SVPEEK will terminate when the end of VDP memory is reached or when the last element of the string array being used is filled.

Example, places the character definition of character 33, defaulted to be the "!", in B$(0).

```
100 CALL LINK("SVPEEK",1032,0,8,A$(),8)
110 CALL LINK("BINHEX",A$(0),B$(0))
```

SVPOKE

String (Master) Video (Display Processor) memory POKE

```
CALL LINK("SVPOKE",NV,NV,NV,SA)
                  R  R  R  R
```

Parm 1: Beginning address in VDP to poke. Range: 0 to 16383.
Parm 2: Bias to add to each byte poked. Range: 0 to 255.
Parm 3: Number of bytes to poke.
Parm 4: String (array) from which to poke.

Comments: See also SVPEEK.

Usage: Pokes VDP memory from a string (array).

Applications: Use to poke color, pattern descriptor, sprite and other table values into VDP.

Cautions: SVPOKE terminates when the end of VDP memory is reached or the last element of an array has been poked, in addition to the standard termination after the specified number of bytes have been poked.

Example, pokes the elements of A$() to fill the Screen Image Table.

```
100 CALL LINK("SVPOKE",0,0,768,A$())
```

TRIM

TRIM of unwanted characters

```
CALL LINK("TRIM",SA,SV)
                 R  R
```

Parm 1: Array to be trimmed.
Parm 2: Character(s) to trim, as a string.

Comments: See also LTRIM.

Usage: Used to eliminate unwanted characters from the right side of a string. For instance, if Parm 2 is "E4", then all "E"'s and "4"'s at the end of each element of the array in Parm 1 will be removed back to the point where a character is not an "E" or a "4".

Applications: Useful for eliminating trailing blanks, carriage returns, line feeds, etc. that might not be desired for a particular situation.

Cautions: TRIM does not eliminate all occurances of the specified character(s), only the occurances at the end of the string. TRIM'ing more than one character at a time should be done with caution, as one of the characters to be trimmed might appear to be safely imbedded within a string until other characters preceding it are eliminated.

Example:

```
100 A$(7)="BEARING"&CHR$(13)&CHR$(10)
110 B$="ING"&CHR$(13)&CHR$(10)
120 CALL LINK("TRIM",A$(),B$
```

WINDOW

screen WINDOW

```
CALL LINK("WINDOW",NV,NV,NV,NV,SV,SA)
                   R  R  R  R  R  R
```

Parm 1: Beginning screen row. Range: 1 to 23.
Parm 2: Beginning screen column. Range: 1 to 31.
Parm 3: Ending screen row, must be larger than Parm 1. Range: 2 to 24.
Parm 4: Ending screen column, must be larger than Parm 2. Range: 2 to 32.
Parm 5: Mode selection, "R" for "READ" a window, "W" for "WRITE" a window. Must be one character.
Parm 6: Array to be windowed.

Comments: See also CIRCUL, FIXLEN, APENDA.

Usage: WINDOW will "read" a rectangular area of the screen and store it in an array or "write" an array to a rectangular area of the screen. The array is written beginning with the first array element, regardless of the beginning screen row. The element is written to the width of the window or to the length of the element, whichever is shorter. Vertically, the window is written to the end of the array or to the bottom row of the window, whichever is lesser. When reading a window, the length is automatically set to the width of the window and continues until the lesser of the end of the window or the last element of the array, with the array being filled beginning at the first element (element zero if the OPTION BASE is zero). Columns are

numbered 1 to 32, as with XB's HCHAR, and rows are numbered 1 to 24.

Applications: WINDOW can often be used as a substitute for DISPLAY AT or PRINT. When combined with other STRING MASTER functions, such as CIRCUL, WINDOW can be very powerful.

Cautions: If WINDOW is used with an array where the elements to be displayed are of a lesser length than the width of the window, then not all of the WINDOW area will be over-written, which is advantageous if the programmer anticipates such a possibility. To clear all of the previous contents of the window, FIXLEN can be used on the array to set the lengths of all elements of the array to the width of the window. If the array must be maintained as it was, then APENDA can be used to copy the array into another array before using FIXLEN.

Example:

```
100 CALL LINK("WINDOW",1,3,5,30,"W",A$())
```

WARRANTY COVERAGE

STRING MASTER is warranted against defective material and workmanship on such materials for a period of 90 days from date of purchase. During the 90 day period, Bytemaster Computer Services will replace any defective products at no additional charge, provided the product is returned, shipping prepaid, to Bytemaster Computer Services. Registration number must be provided with any returns. The Purchaser is responsible for insuring any product so returned and assumes the risk of loss during shipping. This warranty is VOID if the product has been damaged by accident, unreasonable use, neglect, tampering, improper service or other causes not arising out of defects in materials and workmanship.

Ship to:

Bytemaster Computer Services
171 Mustang Street
Sulphur, LA 70663-6724

WARRANTY DISCLAIMERS

Any implied warranties arising out of this sale, including, but not limited to the implied warranties of merchantability and fitness for a particular purpose, are limited in duration to the above 90 day period. Bytemaster Computer Services shall not be liable for loss or use of the software or other incidental or consequential costs, expenses or damages incurred by the consumer or any other user(s).

Some states do not allow the exclusion or limitation of implied warranties or consequential damages, so the above limitations or exclusions may not apply to you in those states.

LEGAL REMEDIES

This warranty gives you specific legal rights and you may also have other rights that vary from state to state.

REPLACEMENT AFTER WARRANTY

After the 90 day warranty period, any original defective diskette may be returned along with a check for $4.00 to cover shipping and diskette costs, and Bytemaster Computer Services will replace it.