STRING MATH SUBROUTINE PACKAGE
STATH 1.2
APRIL 27, 1973

Program Users Guide

Datapoint

STATH Users Guide

Datapoint Corporation

April 27, 1973

INTRODUCTION

Stath is a subroutine package specifically designed to provide formatted keyboard input, screen display, check sum and arithmetic operations on numeric strings. Each function of STATH is obtained by calling the entry point associated with that function.

Following is a list of the functions available through STATH. The labels given to their entry points and the sections incorporating their usage parameters:

ENTRY POINT	FUNCTION
ADD$	Addition
COM$	Compare Magnitude
DIV$	Division
DSP$	Display on screen
KEY$	Keyboard formatted Input
MOD10$	MOD 10 check sum calcuation
MOD11$	MOD 11 check sum calcuation
MOV$	Move string
MUL$	Multiply
SUB$	Subtract

2.0 INTRODUCTION TO STRINGS - NUMERIC AND OTHERWISE

The purpose of a 'string' is to carry around a 'package' of text. A string is an individual block of text and just like a string it has a definite beginning and end. The composition of the string is an uninterrupted sequence of ASCII characters. That is, between the beginning and end of the string and only ASCII characters are allowed.

The string is bounded at the beginning and end in different ways. The end is determined by the first occurrence of the ASCII 'ETX' which is equal to (003) in the sequence of characters called the string. The 003 tells STATH that the string is ended. The CTOS will also accept a carriage return character (015) in place of the 003 but STATH only accepts the 003.

The following are valid strings. The contents of the parentheses are intended to be the byte value of the ASCII character for single character values or the octal value of the octal triple such as 003.

(N) (O) (W) ( ) (I) (S) ( ) (T) (H) (E) ( ) (T) (I) (M) (E) (003)

(0) (1) (2) (3) (4) (5) (6) (7) (8) (9) (003)

Which are in octal:

116,117,127,040,111,123,040,124,110,105,040,124,111,115,105, 003

and

060,061,062,063,064,065,066,067,070,071,003

Although a string has an inherent end built into itself, the 003, there is no beginning. At least no beginning which itself is part of the string of characters in memory. The beginning is combined with the pointer to the string itself. That is, a string is referred to by calling out a location in memory. That location is the first character of the string. In the above samples, for 'now is the time' to be referred to beginning with the word 'now' the location of the letter 'N' would be specified. It is clear that specifying only the 'N' yields a complete description which is:

'Begin with the character in the location specified and continue until a 003 is reached.'

Beginning with 'N' and continuing to the 003 gives: 'Now is the time'. If the location of the letter 'W' in now were specified, the string resulting would be 'w is the time'.

Therefore, to specify a string to a routine (like STATH) which is going to use the string, the user must only transfer the address of the first character of the string or the character in the string the user wants to begin the string (it may not be the first) to the routine. Also, if the user created the string, he must be assured that there is a terminating 003 byte immediately following the last character of the string in memory.

STATH differentiates between two categories of strings:

1) Numeric strings

and

2) Non-numeric strings

Where numeric strings are only regular strings with the character set restricted the characters 0123456789 with an optional single period representing the decimal point and/or a single hyphen leading the string representing a minus sign.

A non-numeric string is any string which is not numeric by the above definition.

A numeric string (ommitting temporarily the 003) can look like:

000000345677888887 77.9999999999991

or

-123.45

or

34.5000000000

There is a size limit as to the number of characters a string may have in STATH. This is not true of ordinary text strings in CTOS where a string may, for some strange purpose, have thousands of characters in it. STATH is a mathematic package and the numeric strings represent numbers. The largest number of digits, therefore, is limited in STATH and that limit is 126.

2.1 INTRODUCTION TO THE FUNCTIONS OF STATH

STATH functions fall in the following four categories. The categories are listed with their appropriate functions below.

ARITHMETIC	ANALYSIS	MANIPULATIVE	INPUT/OUTPUT
Addition	Compare	Move	Keyboard formatted input
Subtraction	MOD10 Check		Display on screen
Division	MOD11 check		
Multiplication			

The arithmetic functions are the normal functions with which everyone is familiar.

The analysis functions permit decisions to be made on the content of a number. MOD10 and MOD11 verify the check sum Modulo 10 or 11 as is used in many business applications. Compare will permit comparing two numbers to determine equality or relative magnitude.

The move function is necessary as a preparation for using the multiplication and division functions in STATH, applicable for general use in the user's program to move numeric strings from one location to another and to format and round them in the process.

The input/output functions provide the user with simple techniques for bringing numbers into memory from the keyboard and displaying string numbers in memory onto the screen.

3.0 STATH FUNCTIONS AND ARGUMENTS

Each routine takes one or two arguments. An argument consists of a CTOS-compatible string. The argument strings are bounded at the end by an ASCII ETX (003), and the beginning boundary is determined by the address contained in the register-pair associated with that argument. The maximum size for any STATH string is 126 characters. This means arguments and results are limited each to 126 digits.

Except for the routine DSP$, all strings must be 'numbers' which means a sequence only of ASCII numeric digits (0123456789) with an optional decimal point, an optional leading minus sign and optional leading blanks (an octal 040). The number must be right justified in the argument string. All strings except for DSP$ set the condition flags as follows:

Flag	Indication
Zero	The result was zero
Sign	The result was negative
Carry	An overflow occured
Parity	One or both arguments were improperly formatted

If parity is not set at the end of an operation, HL and DE contain the addresses of the location in memory past their respective ETX's. In case of KEY$ and DSP$, D contains the column and E contains the row of the position immediately beyond the display area used. MUL$ and DIV$ leave D and E with junk in them. MOD10 and MOD11 leave H and L containing the address of the check digit postion.

3.1 EXAMPLES ON THE USE OF STATH

Following is a 488-byte program which is a useful desk calculator using STATH. It is included as an example of a program calling STATH functions.

'DCLAC', the desk calculator, inputs a numeric string and provides addition, subtraction, multiplication or division of that inputted string against an accumulator. 'DCALC' always inputs the string from the keyboard into a string labeled 'INPUT'. The accumulator is in a string labeled 'ACCUM'.

The four arithmetic operations performed in the program are routines labeled as 'ADDOP', 'SUBOP', 'MULOP', and 'DIVOP'. The routines are very short but demonstrate the use of STATH.

```
        SET     01000
BOOT$   EQU     064
MOV$    EQU     010000
ADD$    EQU     010003
SUB$    EQU     010006
MUL$    EQU     06000
DIV$    EQU     06003
KEY$    EQU     010014
DSP$    EQU     010017
KEYIN$  EQU     017000
DSPLY$  EQU     017151
MLOAD$  EQU     017620
BEEP    EQU     13
HEADING DC      021,011,20,'2200'
        DC      013,5,011,31,'Total'
        DC      013,7,011,28,'Keyboard'
        DC      013,2,011,28,'0 to 9'
        DC      'Decimal Places?'
DECPL   DC      '0',3
OVFMSG  DC      'Overflow',3
BLANK   DC      '        ',3
CLEAR   DC      022,3
INPUT   DC      '0000000000',3
ACCUM   DC      '00000000000',3
DIVID   DC      '0000000000000000000000',3
NAME1   DC      'STATH'
OPCODE  DC      ' ',015
DCALC   DE      NAME1
        CALL    MLOAD$
        JFZ     BOOT$
DCALCH  DE      0
        HL      HEADING
        CALL    DSPLY$
        LD      51
        LE      2
        HL      DECPL
        CALL    KEY$
        LL      INPUT
        CALL    FILLIN
        LL      ACCUM
        CALL    FILLIN
        LL      DIVID
        CALL    FILLIN
        LL      DECPL
        LAM
        SU      '0'
        LBA
        LC      '.'
        LA      INPUT+10
        SUB
        LLA
        LMC
        LA      ACCUM+10
```

```
         SUB
         LLA
         LMC
         LAB
         SLC
         LBA
         LA      DIVID+20
         SUB
         LLA
         LMC
         LD      28
         LE      2
         HL      CLEAR
         CALL    DSPLY$
         DE      ACCUM
         HL      ACCUM
         CALL    SUB$
DCALCL   LD      38
         LE      5
         HL      ACCUM
         CALL    DSP$
         LD      50
         LE      7
         HL      BLANK+6
         CALL    DSP$
         LE      38
         LE      7
         HL      INPUT
         CALL    KEY$
         LC      1
         LE      50
         LE      7
         HL      OPCODE
         CALL    KEYIN$
         HL      OPCODE
         LAM
         CP      015
         JTZ     ADDOP
         CP      'A'
         JTZ     ADDOP
         CP      'S'
         JTZ     SUBOP
         CP      'M'
         JTZ     MULOP
         CP      'D'
         JTZ     DIVOP
         CP      'E'
         JTZ     MOVOP
         CP      'R'
         JTZ     DCALCH
         EX      BEEP
         JMP     DCALCL
ADDOP    DE      INPUT
```

```
        HL      ACCUM
        CALL    ADD$
OVFTST  JFC     NOOVF
        LD      36
        LE      3
        HL      OVFMSG
        CALL    DSP$
        EX      BEEP
        JMP     DCALCL
NOOVF   LE      36
        LE      3
        HL      BLANK
        CALL    DSP$
        JMP     DCALCL
SUBOP   DE      INPUT
        HL      ACCUM
        CALL    SUB$
        JMP     OVFTST
MULOP   DE      ACCUM
        HL      ACCUM
        CALL    MOV$
        DE      INPUT
        HL      ACCUM
        CALL    MUL$
        JMP     OVFTST
MOVOP   DE      INPUT
        HL      ACCUM
        CALL    MOV$
        JMP     OVFTST
DIVOP   DE      ACCUM
        HL      DIVID
        CALL    MOV$
        DE      INPUT
        HL      ACCUM
        CALL    DIV$
        JMP     OVFTST
FILLIN  LAM
        CP      3
        RTZ
        LA      '0'
        LMA
        LAL
        AD      1
        LLA
        JMP     FILLIN
        END     DCALC
```

Observe the addition, 'ADDOP'. To add together the inputted string 'INPUT' to the accumulator 'ACCUM' the user only writes the following code as found at 'ADDOP'.

```
ADDOP   DE      INPUT
        HL      ACCUM
        CALL    ADD$
```

Executing this code will cause string 'INPUT' to be added to the string 'ACCUM' with the result in the string 'ACCUM'. The accumulator, it must be realized, is simply a string which the writer of 'DCALC' is using as his result string and he preferred to call it an accumulator.

Note that after each operation there is a jump to 'OVFTST' or as in 'ADDOP', the code is immediately after and executed right after 'ADDOP'. Observe that the first instruction

```
OVFTST JFC NOOVF
```

of the overflow test is the actual test: If the carry is not set then there was no overflow resulting from the operation. If the carry was set, in 'DCALC' the message 'overflow' is printed on the screen as is seen from the code following the
'JFC NOOVF'.

Subtraction behaves the same as addition except for the CALL to SUB$.

Multiplication and division are slightly different from addition and subtraction but operate similar to each other. Observe the following code as taken from 'DCALC'.

```
MULOP   DE      ACCUM
        HL      ACCUM
        CALL    MOV$
        DE      INPUT
        HL      ACCUM
        CALL    MUL$
        JMP     OVFTST
```

This demonstrates the requirements, as stated in 7.0, that, in MUL$ and DIV$, the argument #2 must be the result of the previous move. The reason for this is that multiplication and division really require three 'registers' or strings: The two strings being multiplied and the result. The 'MOV$' move operation makes a copy of whatever is being moved, during the move, in an internal STATH 'register' string. Therefore, note that the first three instructions in 'MULOP' cause the accumulator to be 'MOV$' moved to itself. Frequently the user can save time by ultilizing this fact in making the last move before calling 'MUL$' a move of a string involving argument #2. (Again, argument #2 is the argument associated with the H and L registers).

Also note that 'MULOP' tests overflow using the same routine that is used for the other three arithmetic routines 'OVFTST' as described above.

4.0 LOADING STATH

STATH may be loaded in memory in either of two ways:

1) Incorporating the source code of STATH into the problem source code.

2) Catalog STATH as an object file and call it in through the operating system.

The second is preferred and simpler, as is done in 'DCALC'. Once cataloged, the following calls STATH into memory:

```
NAME1   DC     'STATH'
        DE     NAME1
        CALL   MLOAD$
```

5.0 ADDITION

Entry Point Name	ADD$
Entry Point Address	10003 Octal
Argument #1 Address	D-E Registers
Argument #2 Address	H-L Registers
Result Location	Argument #2
Arithmetic Function	(Argument #2) = (Argument #2) + (Argument #1)

Action:

Adds two numeric string numbers, rounds, and installs leading blanks and trailing zeros when needed in the result.

Typical calling sequence:

```
ADD$    EQU    010003

        DE     ARG1
        HL     ARG2
        CALL   ADD$
```

Arguments:

Arguments must be each numeric strings of less than 126 characters in length. Argument 1 is addressed by the D and E Registers. Argument 2 is addressed by the H and L Registers and will contain the result.

Result:

The contents of argument 1 (D and E) will remain unchanged.

The contents of argument 2 (H and L) will contain the sum of arguments 2 and 1 and will have leading blanks and trailing zeros when needed.

Changes:

The contents of argument 2 are changed to contain the result.

Errors Recognized:

Improper argument format (parity bit set)
Overflow occurance (carry bit set)

Comparison Flags:

Result was zero (zero bit set)
Result was negative (sign bit set)

6.0 SUBTRACTION

Entry Point Name	SUB$
Entry Point Address	10006 Octal
Argument #1 Address	D-E Registers
Argument #2 Address	H-L Registers
Result Location	Argument #2
Arithmetic Function	(Argument #2) = (Argument #2) - (Argument #1)

Action:

Subtracts one numeric string number from another, rounds and installs leading blanks and trailing zeros when needed in the result.

Typical calling sequence:

```
SUB$    EQU   010006

        DE    ARG1
        HL    ARG2
        CALL  SUB$
```

Arguments:

Arguments must be each numeric strings of less than 126 characters in length. Argument 1 is addressed by the D and E Registers. Argument 2 is addressed by the H and L Registers and will contain the result.

Result:

The contents of argument 1 (D and E) will remain unchanged.

The contents of argument 2 (H and L) will contain the difference of arguments 2 and 1 and will have leading blanks and trailing zeros when needed.

Changes:

The contents of argument 2 are changed to contain the result.

Errors Recognized:

Improper argument format (parity bit set)
Overflow occurrence (carry bit is set)

Comparison Flags:

Result was zero (zero bit is set)
Result was negative (sign bit is set)

7.0 MULTIPLICATION

Entry Point Name	MUL$
Entry Point Address	6000 Octal
Argument #1 Adress	D-E Registers
Argument #2 Adress	H-L Registers
Result Location	Argument #2
Arithmetic Function	(Argument #2) = (Argument #2) X (Argument #1)
Argument Restrictions	Argument #2 must be result of last MOV$ call

Action:

Multiplies two numeric string numbers, rounds and installs leading blanks and trailing zeros when needed in the result.

Typical calling sequence:

```
MUL$    EQU     06000

        DE      ARG2
        HL      ARG2
        CALL    MOV$

        DE      ARG1
        HL      ARG2
        CALL    MUL$
```

Arguments:

Arguments must be each numeric strings of less than 126 characters in length. Argument 1 is addressed by the D and E Registers. Argument 2 is addressed by the H and L Registers and will contain the result. Argument 2 must have been involved in the previous move operation.

Result:

The contents of argument 1 (D and E) will remain unchanged.

The contents of argument 2 (H and L) will contain the product of arguments 2 and 1 and will have leading blanks and trailing zeros when needed.

Changes:

The contents of argument 2 are changed to contain the result.

Errors Recognized:

Improper argument format (parity bit set)
Overflow occurrence (carry bit set)

Comparison Flag:

Result was zero (zero bit set)
Result was negative (sign bit set)

8.0 DIVISION

Entry Point Name	DIV$
Entry Point Address	6003 Octal
Argument #1 Address	D-E Registers
Argument #2 Address	H-L Registers
Result Location	Argument #2
Arithmetic Location	Argument #2) = (Argument #2 / (Argument #1)
Argument Restrictions	Argument #2 must be result of last MOV$ call

Action:

Divides one numeric string number into another, rounds and installs leading blanks and trailing zeros when needed in the result.

Typical calling sequence:

```
MOV$    EQU    010000

        DE     ARG2
        HL     ARG2
        CALL   MOV$

DIV$    EQU    06003

        DE     ARG1
        HL     ARG2
        CALL   DIV$
```

Arguments:

Arguments must be each numeric strings of less than 126 characters in length. Argument 1 is addressed by the D and E Registers. Argument 2 is addressed by the H and L Registers and will contain the result. Argument 2 must have been involved in the previous move operation.

Result:

The contents of argument 1 (D and E) will remain unchanged.

The contents of argument 2 (H and L) will contain the result of the division of argument 1 into argument 2 and will have leading blanks and trailing zeros when needed.

The number of decimal places in the result is equal to the number of decimal places in the dividend less the number of decimal places in the divisor. This number may not be negative and if it is, the number of decimal places is extended to make the difference zero.

The size of the result equals the size of the extended dividend less the size of the divisor.

Note that the string '10.0' divided by the string '3.0' is the string '3'. It is rounded to ZERO decimal places.

Changes:

The contents of argument 2 are changed to contain the result.

Errors Recognized:

Improper argument format (parity bit set)
Overflow occurence (carry bit set)

Comparison Flags:

Result was zero (zero bit set)
Result was negative (sign bit set)

9.0 COMPARE

Entry Point Name	COM$
Entry Point Address	10011 Octal
Argument #1 Address	D-E Registers
Argument #2 Address	H-L Registers
Result Location	Arguments unchanged. Only sets condition code
Arithmetic Function	(cond-code) = (cond [(Argument #2) - (Argument #1)]

Action:

Compares two numeric string numbers as to magnitude. No change to arguments results. Changes are only made to the condition flags.

Typical calling sequences:

```
COM$    EQU    010011

        DE     ARG1
        HL     ARG2
        CALL   COM$
```

Arguments:

Arguments must be each numeric strings of less than 126 characters in length. Argument 1 is addressed by the D and E registers. Argument 2 is addressed by the H and L Registers and will contain the result.

Result:

The contents of both arguments will remain unchanged. Only the condition code will change and will obtain the exact same condition as if a call to SUB$ were done. Therefore, the resultant condition flags will behave as if the result were to be rounded.

Changes:

The contents of both arguments remain unchanged.
Only the condition flags are changed.

Errors Recognized:

Improper argument format (parity bit set)
Overflow occurrence (carry bit set)

Comparison Flags:

Result was zero (zero bit set)
Result was negative (sign bit set)

10.0 MOVE

Entry Point Name	MOV$
Entry Point Address	10000 Octal
Argument #1 Address	D-E Registers
Argument #2 Address	H-L Registers
Result Location	Argument #2
Arithmetic Function	(Argument #2) = (Argument #1)

Action:
Replaces the numeric string number in argument 2 with that of argument 1, rounds and installs leading blanks and trailing zeros when needed in the result.

Typical calling sequence:

```
MOV$    EQU    010000

        DE     ARG1
        HL     ARG2
        CALL   MOV$
```

Arguments:
Arguments must be each numeric strings of less than 126 characters in length. Argument 1 is addressed by the D and E Registers. Argument 2 is adressed by the H and L Registers and will contain the result.

Result:
The contents of argument 1 (D and E) will remain unchanged.
The contents of argument 2 (H and L) will contain the number of argument 1 rounded and reformatted if necessary.

Changes:
The contents of argument 2 are changed to contain the result.

Errors Recognized:
Improper argument format (parity bit set)
Overflow occurence (carry bit set)
[Note that overflow can occur in a MOV$ if a move from a larger to smaller field is attempted]

Comparison Flags:
Result was zero (zero bit set)
Result was negative (sign bit set)

11.0 MOD10 CHECK SUM CLACULATION

Entry Point Name	MOD10$
Entry Point Address	6006 Octal
Argument #1 Address	H-L Registers
Result Location	A-Register (no reformatting of argument)
Arithmetic Function	(A Reg) = Check-MOD-10 (Argument #1)

Action:
Checks validity of Modulo 10 check sum of a numeric string number.

Typical calling sequence:

```
MOD10$  EQU   06006

        HL    ARG1
        CALL  MOD10$
```

Arguments:
The argument must be a numeric string of less than 126 characters in length. Argument 1 is adressed by the H and L Registers.

Result:
The contents of the argument remains unchanged.
The carry bit is set if the check digit is 10.
The zero bit is set if the check digit is not 10.
The check digit is in the A-Register upon return.

Changes:
The contents of the argument remain unchanged.

Errors Recognized:
Improper argument format (parity bit set)

Comparison Flags:
Check digit was 10 (carry bit set)
Check digit was not 10 (zero bit set)

12.0 MOD11 CHECK SUM CALCULATION

Entry Point Name	MOD11$
Entry Point Address	6011 Octal
Argument #1 Address	H-L Registers
Result Location	A-Register (no reformatting of argument)
Arithmetic Function	(A-Reg) = Check-MOD-11 (Argument #1)

Action:
Verifies the Modulo 11 check sum of the numeric string number.

Typical calling sequence:

```
MOD11$  EQU   06011

        HL    ARG1
        CALL  MOD11$
```

Arguments:
The argument must be a numeric string of less than 126 characters in length. The argument is addressed by the H and L Registers.

Result:
The contents of the argument remains unchanged.
The carry bit is set if the check digit is 11.
The zero bit is set if the check digit is not 11.
The A-Register contains the check digit.

Changes:
The contents of the argument remain unchanged.

Errors Recognized:
Improper argument format (parity bit set)

Comparison Flags:
Check digit was 11 (carry bit set)
Check digit was not 11 (zero bit set)

13.0 KEYBOARD FORMATTED INPUT

Entry Point Name	KEY$
Entry Point Address	10014 Octal
Argument #1 Address	H-L Registers
Extra Parameters	(D Reg) = Column. (E Reg) = Row for cursor
Input Function	(Argument #1) = (Keyed in number)
Input Restrictions	Screen format and, therefore, keyed in number has same format as originally in Argument #1

Action:

Provides formatted input from the keyboard into a numeric string. The format is maintained on the screen and only a number fitting the format can be entered. The input numeric string is placed in argument 1.

Typical calling sequence:

```
KEY$    EQU    010014

        LD     COLUMN
        LE     ROW
        HL     ARG1
        CALL   KEY$
```

Arguments:

The argument must be a formatted numeric string.

The D and E Registers must contain the column and row of the cursor position of the first character to be typed in.

Result:

The contents of argument 1 are replaced by the input number.

Striking the enter key with no input will cause the argument to be replaced with a zero.

The H and L Registers are pointing immediately after the ETX.

Changes:

The contents of the argument are replaced with the input string.

Errors Recognized:

Improper argument format (parity bit set)

Comparison Flags:

Result was zero (zero bit set)

Result was negative (sign bit set)

14.0 DISPLAY STRING

Entry Point Name	DSP$
Entry Point Address	10017 Octal
Argument #1 Address	H-L Registers
Extra Parameters	(D Reg) = Column. (E Reg) = Row for cursor
Input Functions	(Display starting at D,E) = (Argument #1)
Input Restrictions	None. May even be non-numeric string

Action:

Displays a string onto the screen. String may be non-numeric.

Typical calling sequence:

```
DSP$    EQU    010017

        LD     COLUMN
        LE     ROW
        HL     ARG1
        CALL   DSP$
```

Arguments:

The argument may be a numeric or non-numeric string as long as it terminates with a ETX. The D and E Registers contain the column and row of the location of the first character of the string.

Result:

The string in argument 1 is displayed on the screen starting at the cursor location beginning with the column and row specified by the D and E Registers.

The H and L Registers point the location immediately after the ETX in argument 1.

Changes:

The contents of the argument remain unchanged.

Errors Recognized:

None

Comparison Flags:

None