MPCNC Forth Controller Userguide

Version 0.6

1. Introduction

MPCNC Forth Controller allows users to write programs to interactively control the MPCNC (or any Marlin style software)

1.1. A Forth Like Language

While the programming language and environment is base on Forth, there are several differences. Of Particular interest are the following:

1) Data types. Forth historically is not data typed. For example, pushing a string on the stack results in a series of bytes representing the characters of the string appear on the stack.

   Unlike traditional Forth, MPCNC Forth has two different data types. Numeric, and string.

   - Numeric data:Floating point numbers.
   - String data: normal C# style string.

   With MPCNC Forth attempting a operation that expects a number on a string, and vice versa, will generate an error. Restricting free access between data types help reduce unexpected operation from programming errors, and is only limiting to the most advanced programmers.

2) String Storage on Stack. Strings are stored on the stack as a c# string as part of the stack entry class, not as an address/count as standard Forth.

3) Serial Port Extension. The addition of specific words and features allows for MPCNC Forth Controller to communicate with a Marlin (or similar) controller.

4)

2. Word Summary

A note about word descriptions. Each word gives a quick reference for how it interacts with the stack within parentheses.

(a b | — sum(a+b))

In the example above, the stack condition before the word is executed is shown in the first part of the notation (the part before the vertical bar. The part after the vertical bar indicates the state of the stack after execution. The character "—" represents no value, or at least no value relevant to the execution of the given word. So in the example above, (for the word "+") expected two numbers on the stack, a and b. The word takes these two values off the stack and puts their sum back on the stack.

2.1. Common Forth Words

2.1.1. CR (— — | — —)

Expects on stack: Nothing
Returns on stack: Nothing
Operand Type: none
Description: Simply outputs a Carriage Return to Console Output

---

2.1.2. SPACE (— — | — —)

Expects on stack: Nothing
Returns on stack: Nothing
Operand Type: none
Description: Simply outputs a space to Console Output

2.1.3. SPACES (— n | — —)

Expects on stack: count
Returns on stack: Nothing
Operand Type: numeric
Description: Simply outputs "n" spaces to Console Output

2.1.4. EMIT (— n | — —)

Expects on stack: Number representing an Ascii character
Returns on stack: Nothing
Operand Type: numeric
Description: Outputs the represented Ascii character to the console

2.1.5. ." <string "> dot-quote (— — | — —)

Expects on stack: Nothing
Returns on stack: Nothing
Operand Type: None
Description: Parses a string following the ." to another " in the word stream. String between is output to the console.

2.1.6. S" <string "> S-quote (— — | — string)

Expects on stack: Nothing
Returns on stack: String
Operand Type: String
Description: Parses a string following the ." to another " in the word stream. String between is output to the console.

2.1.7. . dot (— number or string | — —)

Expects on stack: a number or a string
Returns on stack: nothing
Operand Type: Number or String
Description: Outputs the variable on the stack to the console, regardless of type.

2.1.8. + plus (n1 n2 | —(n1+n2))

Expects on stack: two numbers
Returns on stack: the sum of the two numbers
Operand Type: numeric
Description: Pops two stack entries. Errors if stack entries are not both numeric. Returns the sum of the two numbers to the stack.

2.1.9. - minus (n1 n2 | — (n1-n2))

Expects on stack: two numbers
Returns on stack: the difference of the two numbers
Operand Type: numeric
Description: Pops two stack entries. Errors if stack entries are not both numeric. Returns the difference of the two numbers to the stack.

2.1.10. * multiply (n1 n2 | — (n1*n2))

Expects on stack: two numbers
Returns on stack: the difference of the two numbers
Operand Type: numeric
Description: Pops two stack entries. Errors if stack entries are not both numeric. Returns the product of the two numbers to the stack.

2.1.11. / divide (n1 n2 | — (n1/n2))

Expects on stack: two numbers
Returns on stack: the difference of the two numbers
Operand Type: numeric
Description: Pops two stack entries. Errors if stack entries are not both numeric. Returns the quotient of the two numbers to the stack.

2.1.12. SWAP (n1 n2 | n2 n1)

Expects on stack: two values
Returns on stack: two values
Operand Type: numeric or string
Description: exchanges the position of the top two items on the stack.

2.1.13. DUP (— n1 | n1 n1)

Expects on stack: one value
Returns on stack: two of the same value
Operand Type: numeric or string
Description: duplicates the last item on the stack.

2.1.14. OVER (n1 n2 | n2 n1 n2)

Expects on stack: two values
Returns on stack: three values
Operand Type: numeric or string
Description: copies the second item on the stack and pushes it onto the stack.

2.1.15. ROT (n1 n2 n3 | n2 n3 n1)

Expects on stack: three values
Returns on stack: three values
Operand Type: numeric or string
Description: pulls the third entry from the stack and pushes it on top.

2.1.16. DROP (— n1 | — —)

Expects on stack: one value
Returns on stack: Nothing
Operand Type: numeric or string
Description: Pops the top item from the stack, and discards it.

2.1.17. DICT (— — | — —)

Expects on stack: Nothing
Returns on stack: Nothing
Operand Type: N/A
Description: Dumps the entire dictionary to the console

2.1.18. : COLON <word> (starts word definition) (— — | — —)

Expects on stack: Nothing
Returns on stack: Nothing
Operand Type: N/A
Description: Starts to compile a new Dictionary entry. Colon must be followed by a word that will be the name of the new function in the dictionary. A Semicolon ends the definition.

2.1.19. ; SEMICOLON (ends word definition) (— — | — —)

Expects on stack: Nothing
Returns on stack: Nothing
Operand Type: N/A
Description: Ends a new word definition started by : (COLON).

2.1.20. ‘ TICK <word> (— — | — a1)

Expects on stack: Nothing
Returns on stack: Address of word following tick
Operand Type: numeric
Description: pushes the address (index) of the word onto the stack. If word is not defined, will push -1 onto stack
Usage:

EXAMPLE 1: The routine ‘bob’ is undefined

```
ok> ‘ bob . {enter}
-1
ok >
```

EXAMPLE 2: get the index of the built in code DUP

```
ok> ‘ dup .
13
ok >
```

2.1.21. SEE <word> (— — | — —)

Expects on stack: Nothing
Returns on stack: Nothing
Operand Type: N/A
Description: If WORD is not a built-in word, it will be decompiled.

Example 1:

```
ok >: bob 1 2 + . ;
ok >see bob
: bob
  1 2 + .
;

ok>
```

2.1.22. CODE (— — | — —)

Expects on stack: Nothing
Returns on stack: Nothing
Operand Type: N/A
Description: Dumps codespace. Each entr in codespace is made up of code space class.

```
ok > : bob 1 2 + . ;

ok >code
Code Dump
Index:0  Address:21  Flags:1  String:  Value:-1
Index:1  Address:23  Flags:1  String:  Value:1
Index:2  Address:23  Flags:1  String:  Value:2
Index:3  Address:8  Flags:1  String:  Value:-1
Index:4  Address:7  Flags:1  String:  Value:-1
Index:5  Address:22  Flags:1  String:  Value:-1

ok >
```

2.1.23. STACK (— — | — —)

Expects on stack: Nothing
Returns on stack: Nothing
Operand Type: N/A
Description: Nondestructively displays the stack in detail, including decoding of flag field.

2.2. Commenting in Code

There are several ways of commenting in code.

- For word definitions use “(“. However, you can actually put any text you’d line within the open and close parentheses.

```
EXAMPLE:
(    – X; | – -X  )
( This is a comment )
( So is this)

( this is a comment )
```

- For commenting to the end of line, the c++ comment style is accepted “//“

```
EXAMPLE:

23 DUP DROP POP // this code does nothing useful

// this entire line is a comment
```

- For lengthy comments, or for temporarily commenting out code use the c style comment /* */

```
EXAMPLE:

/* this is a comment */
/* this
      Is
       Also
         A
          Comment */
```

2.3. Forth Constants and Variables

2.3.1.CONSTANT <word> (— n1 | — —)

Expects on stack: A value for the new constant
Returns on stack: Nothing
Operand Type: numeric or string
Description: Defines <word> in dictionary as constant and assigns it the value n1
Execution: Places the value of constant on the stack

EXAMPLE 1:
```
ok> 123 constant bob
ok > bob .
123
```

EXAMPLE 2: (constants cannot be assigned)
```
ok >0 bob !
Error Executing Primitive: Address on stack does not represent the
address of a variable
ok >
```

2.3.2. VARIABLE <word> (— — | — —)

Expects on stack: Nothing
Returns on stack: Nothing
Operand Type: numeric or string
Description: Defines <word> in dictionary as a variable
Execution: Places the index of the variable on the stack
Notes: Variable is initialized as the number zero. However you can store either numbers or strings to the variable, and change the type at any time by setting a new value.

EXAMPLE1: Define the variable
```
ok > variable bob
ok >
```

EXAMPLE 2: Get the index of bob .
```
ok > bob .
59
ok >
```

Example 3: Store and Fetch variable
```
ok >123 bob !
ok >bob @
123
ok >
```

2.3.3. @ (fetch) (— (variable index) | — (variable value))

Expects on stack: index of variable
Returns on stack: Contents of variable
Operand Type: numeric or string
Description: Used to “fetch” the value of a variable
Notes: Variables can be strings or numbers. You can change the type of the variable by storing the desired type to it.

See examples under VARIABLE

2.3.4. ! (store) ((value1) (variable index) | — —)

Expects on stack: value to store in variable and index of variable
Returns on stack: Nothing
Operand Type: numeric or string
Description: Used to “Store” a value to a variable
Notes: Variables can be strings or numbers. You can change the type of the variable by storing the desired type to it.

See examples under VARIABLE

2.3.5. ? (fetch and print) (— (variable index) | — —)

Expects on stack: index of variable
Returns on stack: nothing
Operand Type: numeric or string
Description: Used to “fetch” and print the value of a variable to the console.
Essentially equivalent to @(fetch) .(dot)
Notes: Variables can be strings or numbers. You can change the type of the variable by storing the desired type to it.

2.4. Forth Bools

Forth interprets a regular number as a bool. A value of ZERO is FALSE, and a value of NON-ZERO is TRUE. Negative 1 (-1) is normally used as true, but any non-zero number will be interpreted as TRUE

2.4.1. TRUE (— — | — -1)

Expects on stack: Nothing
Returns on stack: -1
Operand Type: numeric
Description: Pushes a negative one onto the stack, although any non-zero number will be interpreted as true.

2.4.2. FALSE (— — | — 0)

Expects on stack: Nothing
Returns on stack: 0
Operand Type: numeric
Description: Pushes a ZERO onto the stack. Zero is interpreted as false.
, although any non-zero number will be interpreted as true.

2.4.3. INVERT (— n1 | — !n1)

Expects on stack: one value
Returns on stack: inversion of value
Operand Type: numeric only
Description: If n1 = 0 (false) then !n1 = -1 (true)
if n1 <> 0 (note false) then !n1 = 0 (false)

2.5. Forth Boolean Operators

Forth interprets a regular number as a bool. A value of ZERO is FALSE, and a value of NON-ZERO is TRUE. Negative 1 (-1) is normally used as true, but any non-zero number will be interpreted as TRUE.

However, Boolean operations can be applied to any number.

2.5.1. AND (n1 n2 | — (n1 AND n2))

Expects on stack: two numbers
Returns on stack: ANDED value of the two given numbers
Operand Type: numeric
Description: Performs Bitwise AND of n1 and n2 and pushes the result on the stack

Example:

```
n1 = 10  = binary   1    0    1    0
n2 = 12  = binary   1    1    0    0
             AND   ———————————————-
result =            1    0    0    0
```

2.5.2. OR (n1 n2 | — (n1 OR n2))

Expects on stack: two numbers
Returns on stack: OR'ed value of the two numbers given.
Operand Type: numeric
Description: Performs Bitwise AND of n1 and n2 and pushes the result on the stack

Example:

```
n1 = 10  = binary   1    0    1    0
n2 = 12  = binary   1    1    0    0
              OR   ———————————————-
result =            1    1    1    0
```

2.5.3. XOR (n1 n2 | — (n1 OR n2))

Expects on stack: two numbers
Returns on stack: OR'ed value of the two numbers given.
Operand Type: numeric
Description: Performs Bitwise XOR of n1 and n2 and pushes the result on the stack. Where a given bit is the same in both numbers, a 0 bit results. If a given bit is different between the numbers, a 1 bit results

Example:

```
n1 = 10  = binary   1    0    1    0
n2 = 12  = binary   1    1    0    0
              OR   ———————————————-
result =            0    1    1    0
```

2.6. Forth Comparators

Comparators push a true or false onto the stack based on the comparison of two numbers

---

2.6.1. = (n1 n2 | — (true/false))

Expects on stack: two numbers
Returns on stack: True if n1=n2, False if n1<>n2
Operand Type: numeric
Description: Pushes a true or false on the stack depending on the result of the comparison

---

2.6.2. < (n1 n2 | — (true/false))

Expects on stack: two numbers
Returns on stack: True if n1<n2, False if n1>=n2
Operand Type: numeric
Description: Pushes a true or false on the stack depending on the result of the comparison

---

2.6.3. > (n1 n2 | — (true/false))

Expects on stack: two numbers
Returns on stack: True if n1>n2, False if n1<=n2
Operand Type: numeric
Description: Pushes a true or false on the stack depending on the result of the comparison

---

2.6.4. <= (n1 n2 | — (true/false))

Expects on stack: two numbers
Returns on stack: True if n1<=n2, False if n1>n2
Operand Type: numeric
Description: Pushes a true or false on the stack depending on the result of the comparison

---

2.6.5. >= (n1 n2 | — (true/false))

Expects on stack: two numbers
Returns on stack: True if n1>=n2, False if n1<n2
Operand Type: numeric
Description: Pushes a true or false on the stack depending on the result of the comparison

2.7. Forth Zero Comparators

Zero comparators push a true or false onto the stack based on the comparison of one number and zero

---

2.7.1. 0= (— n1 | — (true/false))

Expects on stack: One number
Returns on stack: True if n1=0, False if n1<>0
Operand Type: numeric
Description: Pushes a true or false on the stack depending on the result of the comparison

---

2.7.2. 0< (— n1 | — (true/false))

Expects on stack: One number
Returns on stack: True if n1<0, False if n1>=0
Operand Type: numeric
Description: Pushes a true or false on the stack depending on the result of the comparison

---

2.7.3. 0> (— n1 | — (true/false))

Expects on stack: One number
Returns on stack: True if n1>0, False if n1<=0
Operand Type: numeric
Description: Pushes a true or false on the stack depending on the result of the comparison

2.8. Forth Conditional Execution

Conditional execution is achieved with the words IF, ELSE, and ENDIF.
- IF expects a value on the stack, which interprets as a boolean (0 = false, not 0 = true)
- When IF finds a false, it will jump to ELSE (if there is one) or to ENDIF, if there is no else.
- When IF finds a true, it will execute the following statements until it finds an ELSE or ENDIF.
- When an ELSE is encounter, execute jumps to ENDIF.

Conditionals can be nested.

Example 1 : (— false | — —)

```
IF
    <statements>  // SKIPPED
ENDIF
```

When IF encounters a FALSE on the stack, it will skip statements and continue executing at ENDIF.

Example 2 : (— true | — —)

```
IF
    <statements>  // EXECUTED
ENDIF
```

When IF encounters a TRUE on the stack, it will execute the statements and continuing through the ENDIF.

IF, ELSE, ENDIF examples continued…

Example 3 : (— false | — —)

```
IF
    <statements>  // SKIPPED
ELSE
    <other statements> // executed
ENDIF
```

When IF encounters a FALSE on the stack, it will skip statements and continue executing at ELSE.

Example 2 : (— true | — —)

```
IF
    <statements>  // EXECUTED
ELSE
     <other statements> // SKIPPED
ENDIF
```

When IF encounters a TRUE on the stack, it will execute the statements and continuing up to the ELSE and then jump to ENDIF.

2.9. Forth Looping Constructs

Two looping forms are currently supported b MPMCNC Forth Controller.

They are:

DO….LOOP
BEGIN….UNTIL

Note that the words WHILE and LEAVE are not supported in version 1.x of the MPCNC Forth Controller.

DO (Limit Count | Limit Count)
Expects on stack: two numbers, the limit and the starting count.
Returns on stack: Stack is unchanged
Operand Type: numeric
Description: Do, actually does nothing, it's just represents the point to loop back to in the code.

2.9.1. LOOP (Limit Count | Limit Count+1)

Expects on stack: two numbers, the limit and the current count.
Returns on stack: Two numbers, the limit and the updated count.
Operand Type: numeric
Description: Loop increments the count by one. If the count is less than the limit, execution will jump back to the location of DO. If the count, once incremented, matches or exceeds the limit, the count and limit values are dropped off the stack and execution continues following the LOOP statement.

Example1:

```
10 0
  DO
        <statements>  // will be executed nine times.
  LOOP
```

2.9.2. BEGIN (— — | — —)

Expects on stack: nothing
Returns on stack: Stack is unchanged
Operand Type: N/A
Description: BEGIN, actually does nothing, it's just represents the point to loop back to in the code.

2.9.3. UNTIL (— flag | — —)

Expects on stack: One number (flag) which it will be interpreted as a boolean
Returns on stack: flag removed
Operand Type: numeric/boolean
Description: UNTIL tests the value of flag, and if false, execution continues at the BEGIN statement. Once the flag is TRUE, execution will drop through the UNTIL statement.

Example1:

```
BEGIN
      <statements>  // will be executed until flag is true.
  push a flag to test
UNTIL
```

2.9.4. I,J,K (— — | — index)

Expects on stack: The construct of a DO word.
Returns on stack: integer value
Operand Type: numeric
Description: Gets the iterator for the DO loop.
I = inside most loop
J = second loop nested
K = third loop nested

Example1:

```
 : bob 3 0 DO i . LOOP ;

ok>bob
0 1 2
ok>
```

2.10. Math Functions

2.10.1. SIN (— angleInDegrees | — SIN(a))

Expects on stack: A number representing degrees of an angle
Returns on stack: Sin function of the angle
Operand Type: numeric
Description: Gets the SIN of an angle

2.10.2. COS (— angleInDegrees | — COS(a))

Expects on stack: A number representing degrees of an angle
Returns on stack: COSIN function of the angle
Operand Type: numeric
Description: Gets the COS of an angle

2.10.3. TAN (— angleInDegrees | — TAN(a))

Expects on stack: A number representing degrees of an angle
Returns on stack: Tangent function of the angle
Operand Type: numeric
Description: Gets the TAN of an angle

2.10.4. RND (— — | — value)

Expects on stack: nothing
Returns on stack: A random floating point value between 0 and 1
Operand Type: numeric
Description: Generates a random number

2.10.5. RANGE (max min | — value)

Expects on stack: nothing
Returns on stack: A random floating point value between min and max values.
Operand Type: numeric
Description: Generates a random number within the specified range.

2.10.6. INT (— float | — int)

Expects on stack: a floating point value
Returns on stack: the integer portion of the passed value
Operand Type: numeric
Description: Returns the integer of a given number

2.10.7. ABS (— num | — abs(num))

Expects on stack: a floating point value
Returns on stack: the absolute value of the passed value
Operand Type: numeric
Description: Returns the the absolute value of a given number

2.10.8. MIN (N1 N2 | — N)

Expects on stack: two floating point values
Returns on stack: the lesser of the values passed
Operand Type: numeric
Description: Returns the lesser of the values passed.

2.10.9. MAX (N1 N2 | — N)

Expects on stack: two floating point values
Returns on stack: the greater of the values passed
Operand Type: numeric
Description: Returns the greater of the values passed.

2.10.10. EXP (— N | — e^N)

Expects on stack: a number
Returns on stack: e raised to the power of the supplied number
Operand Type: numeric
Description: Returns e to the power of N

2.11. Miscellaneous Functions

2.11.1. WAIT (— value | — —)

Expects on stack: One number indicating how many milliseconds to wait
Returns on stack: none
Operand Type: numeric
Description: Systems will wait number of milliseconds.

2.11.2. SCREENSHOT (— filename | — —)

Expects on stack: filename as a string
Returns on stack: Nothing
Operand Type: N/A
Description: Provides a way to capture a screenshot

Example:

2.12. String Functions

Note that this implementation of forth stores a string as a c# string, not as a series of bytes on the stack. Because of this, special words to do basic functions are not needed. SWAP, DUP, OVER, ROT, DROP will work equally well on numbers or strings. Since implementations of Forth String support seem to be quite variable. The following words may not match other forth implementations. These are not zero terminated strings.

2.12.1. LEN (— S1 | — N1)

Expects on stack: A string
Returns on stack: a number representing the length of the string
Operand Type: String
Description: Get the length of a given string

2.12.2. CONCAT (S2 S1 | — S2+S1)

Expects on stack: Two string
Returns on stack: A string starting with S2 and ending in S1|
Operand Type: Strings
Description: Concatenate strings

2.12.3. CONTAINS (S2 S1 | — true/false)

Expects on stack: two strings
Returns on stack: -1 if S1 is found in S1. Not found = 0
Operand Type: strings
Description: Find any occurrence of S1 in S2 and return true or false

---

```
 COMPARE ( a1 a2 n -- status )

: SAME? ( a1 a2 n -- t | f )

: CFIND ( c a1 n -- a2 | 0 )                    \ "c-find"

: CFIND< ( c a1 n -- a2 | 0 )                   \ "c-find-
back"

: $LIT ( -- string )                            \ "s-lit"
```

```
: $, ( string -- )                              \ "s-comma"

: ,$ ( -- )                                     \ "comma-s"

: [$LIT] ( -- string )                          \ "bracket-s-
lit"

: $KEY ( string -- )                            \ "s-key"

: $$KEY ( string -- )                           \ "s-s-key"

: $ALLOT ( number -- )                          \ "s-allot"

: $IMPORT ( addr length string -- )             \ "s-import"

: $EXPORT ( string addr -- )                    \ "s-export"


: $. ( string -- )                              \ "s-dot"

: $NULL ( string -- )                           \ "s-null"

: $$+ ( string1 string2 -- )                    \ "s-s-plus"

: $C! ( char string index -- )                  \ "s-c-store"

: $C@ ( string index -- char )                  \ "s-c-fetch"

: $$! ( string1 string2 -- )                    \ "s-s-store"

: $$@ ( string1 string2 index length --)        \ "s-s-fetch"

: $CINS ( char string index -- )                \ "s-c-ins"

: $$INS ( string1 string2 index -- )            \ "s-s-ins"

: $|TRIM  ( string number -- )                  \ "s-left-
trim"
```

```
: $TRIM|  ( string number -- )                \ "s-trim-
right"

: $|SPACES ( string -- )                      \ "s-left-
spaces"

: $SPACES| ( string -- )                      \ "s-spaces-
right"

: $DEL ( string index number -- )             \ "s-del"

: $$REP ( string1 string2 index -- )          \ "s-s-rep"

: $<ROT ( string -- )                         \ "s-left-
rote"

: $>ROT ( string -- )                         \ "s-right-
rote"

: $<<ROT ( string number -- )                 \ "s-many-
left-rote"

: $>>ROT ( string number -- )                 \ "s-many-
right-rote"

: $$COMPARE ( string1 string2 -- status )     \ "s-s-
compare"

: $$= ( string1 string2 -- t | f )            \ "s-s-equal"

: $$< ( string1 string2 -- t | f )            \ "s-s-less-
than"

: $$<= ( string1 string2 -- t | f )           \ "s-s-less-
than-or-equal"

: $$> ( string1 string2 -- t | f )            \ "s-s-
greater-than"
```

```
: $$>= ( string1 string2 -- t | f)           \ "s-s-
greater-than-or-equal"

: $$<> ( string1 string2 -- t | f )          \ "s-s-not-
equal"

: $CFIND ( char string -- index | -1 )       \ "s-c-find"

: $CFIND< ( char string -- index | -1 )      \ "s-c-find-
back"

: $$FIND ( string1 string2 index length -- index | -1 )
                                             \ "s-s-find"

: $CMEM ( char string -- t | f )             \ "s-c-mem"

: $$VER ( string1 string2 -- index | -1 )    \ "s-s-ver"

: $>UPPER ( string -- )                      \ "s-to-upper"

: $>LOWER ( string -- )                      \ "s-to-lower"

: $CONVERT ( string index -- )               \ "s-convert"

: $<N ( number -- )                          \ "s-from-n"

: $>N ( -- number | -1 )                     \ "s-to-n"
```

2.13. GCODE Basic Extensions

```
        DictionaryAdd ("SEND", 55, flagPrim, "", 0f); // if
        DictionaryAdd ("SENDWAIT", 56, flagPrim, "", 0f);

DictionaryAdd ("SETLASERPOWER", 70, flagPrim, "", 0f); //
```

2.13.1. LASERON (Power | — —)

Expects on stack: A number from 0-255 for laser power
Returns on stack: Nothing
Operand Type: Numeric
Description: Set laser on to power value passed. Uses LASERCOMMANDPREFIX

2.13.2. LASERLOFF (— — | — —)

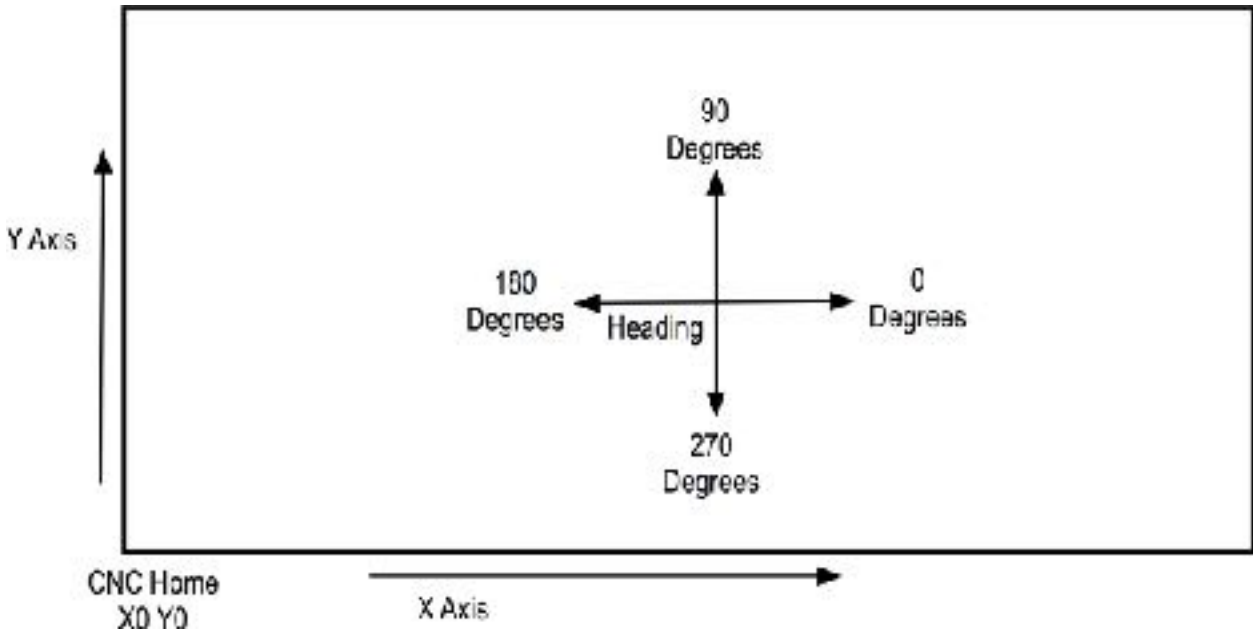
Expects on stack: nothing
Returns on stack: nothing
Operand Type: N/A
Description: Sends power of zero using LASERCOMMANDPREFIX

2.13.3. HOME (— — | — —)

Expects on stack: nothing
Returns on stack: nothing
Operand Type: N/A
Description: Sends home command - G28

2.14. Turtle Graphic Functions for the MPCNC

Vocabulary is based on the vocabulary of : FORTH TURTLE GRAPHICS PLUS by Willian Volk



2.14.1. TURTLEINIT (— — | — —)

Expects on stack: nothing
Returns on stack: Nothing
Operand Type: N/A
Description: Initializes turtle for movement.

MUST BE CALLED ONCE BEFORE ANY OTHER TURTLE COMMANDS

Expects CNC machine to be in initial position.
- If PEN/MARKER, pen must be at Z0 or touching the surface of the target
- if LASER, laser must be positioned and focused
- if ZenXY, doesn't matter
- if ROUTER, router must be at Z0 or touching the surface of the target

What this command does:
- if PEN/MARKER, raises Z by PENSTANDOFFDISTANCE
- if LASER, issues laser off command
- if ZenXY, X and Y will be homed.
- if ROUTER, Raises Z by ROUTERSTANDOFFDISTANCE

2.14.2. MOVE (— dist | — —)

Expects on stack: Distance to move in mm
Returns on stack: Nothing
Operand Type: Numeric
Description: Moves turtle forward in the direction it is facing for distance set in mm if pen is down it will draw. If pen is up, no line will be made.

Notes: Be sure to have set the machine to relative mode, G91, or SETRELATIVEMODE
Also you may wish to set an XY federate XYFEEDRATE.

2.14.3. TURN (— relative-angle | — —)

Expects on stack: Change direction by the angle given.
Returns on stack: Nothing
Operand Type: Numeric
Description: Changes the direct by the angle given. Values may be negative. No movement or drawing occurs with this command.

Example: Current angle = 90

10 TURN

Angle changed to 100.

2.14.4. TURNTO (— absolute-angle | — —)

Expects on stack: Set the direction to the angle given.
Returns on stack: Nothing
Operand Type: Numeric
Description: Sets the new angle specified No movement or drawing occurs with this command.

Example: Current angle = 90

10 TURNTO

Angle changed to 10.

2.14.5. PEN (— — | — true/false)

Expects on stack: nothing
Returns on stack: True (-1) if pen is down, else False (0)
Operand Type: Numeric
Description: Returns state of pen (down or up)

2.14.6. PENDOWN (— — | — —)

Expects on stack: nothing
Returns on stack: nothing
Operand Type: N/A
Description: Sets the pen state to down

Sets internal pen state to DOWN.

Note, this doesn't actually make any movement or send commands.
In the PENDOWN state, the pen/router will be lowered only when there is a movement command performed. (This keeps the pen from leaking, or the laser burning).

2.14.7. PENUP (— — | — —)

Expects on stack: nothing
Returns on stack: nothing
Operand Type: N/A
Description: Sets the pen state to up

Sets internal pen state to up.

Note, this doesn't actually make any movement or send commands.
Pen/router is normally in the up state on the machine, and the laser is off. It is only put down/turned on when a movement command is issued and a PENDOWN has been issued.

2.14.8. ABSOLUTE (— — | — —)

Expects on stack: nothing
Returns on stack: nothing
Operand Type: N/A
Description: Sends a G90 command and waits for response.

2.14.9. RELATIVE (— — | — —)

Expects on stack: nothing
Returns on stack: nothing
Operand Type: N/A
Description: Sends a G91 command and waits for response.

2.14.10. CIRCLECW (I J | — —)

Expects on stack: Values for I and J for G02 command
Returns on stack: nothing
Operand Type: N/A
Description: Draws a circle starting at current position based on I and J for a G02 command
Aliases: Circle

2.14.11. CIRCLECCW (I J | — —)

Expects on stack: Values for I and J for G03 command
Returns on stack: nothing
Operand Type: N/A
Description: Draws a circle starting at current position based on I and J for a G03 command

2.14.12. MOVETO (X Y | — —)

Expects on stack: X and Y locations to move to
Returns on stack: Nothing
Operand Type: Numeric
Description: Moves turtle forward in the direction it is facing for distance set in mm
if pen is down it will draw. If pen is up, no line will be made.

Notes: Consider if you want relative mode or absolute mode for this command.
Also you may wish to set an XY federate XYFEEDRATE.

2.14.13. SPIRAL (CenterX CenterY, StartRadius, EndRadius, Direction, Spacing, StartAngleinDegrees Distancebump | --)

Expects on stack: CenterX - starting x location
CenterY - starting y location
StartRadius - beginning radius
EndRadius - ending radius
Direction - 1=CW -1= CCW
Spacing - how far each loop is spaced
StartAngleInDegrees - position in degrees of start
DistanceBump - modifier for Spacing

Returns on stack: Nothing
Operand Type: Numeric
Description: Draws a spiral
if pen is down it will draw. If pen is up, no line will be made.

Notes: CenterX and CenterY are absolute values
Also you may wish to set an XY federate XYFEEDRATE.

Here's some example code:

```
Turtleinit
Pendown

( Declare Variables )
Variable CenterX    300 CenterX !
Variable CenterY    150 CenterY !
Variable StartRadius  10 StartRadius  !
Variable EndRadius   100 EndRadius  !
Variable Direction   1 Direction  !
Variable Spacing    1  Spacing  !
Variable StartAngle  180 StartAngle  !
Variable DistanceBump  0.005 DistanceBump !

( Put variables on Stack )
CenterX @
CenterY @
StartRadius @
EndRadius @
Direction @
Spacing @
StartAngle @
DistanceBump @

( call function )
 Spiral
```

Example Output for Code as Shown.

2.14.14. L-System Words

Word set to create an manipulate L-Systems

2.14.14.1. LSYSTEM (— seed | — id)

Expects on stack: A string containing the seed for the L-Systrem or axiom
Returns on stack: id of the new L-System to be used to further reference and manipulate it.
Operand Type: N/A
Description: Creates a new L-System and gives an ID to reference it in the future

Example: Creates an L-System with an axiom of “F”, and storing the resultant reference id in variable myLsystem.

```
VARIABLE MYLSYSTEM
// create LSYTEM
s" F " LSYSTEM   MYLSYSTEM !
```

2.14.14.2.LSYSTEM_ADDRULE (“key” “value” id | — —)

Expects on stack: Two strings, and a value. A single letter string representing the “key” for the rule, and a string for substitution for the key. An L-System ID must be included.
Returns on stack: nothing
Operand Type: Two strings an an L-System id
Description: Adds a rule to the dictionary for the referenced L-System.

Example:

Given the variable MYLSYSTEM contains the id of the L-system being used.

```
// ADD RULES
S" F "   S"  F+F-F-F+F "  myLsystem @  LSYSTEM_ADDRULE
```

2.14.14.3. LSYSTEM_SUBSTITUTE (— id | — —)

Expects on stack: An L-system ID
Returns on stack: nothing
Operand Type: An L-System id
Description: makes one pass through the current working string making substitutions according to the rules added in the database.

2.14.14.4. LSYSTEM_PREPARE (-- initalDirection, initialSize, initialAngle, anglegrowth, sizegrowth, startX startY LsystemIndex | --)"

Expects on stack: An L-system ID
Returns on stack: nothing
Operand Type: An L-System id
Description: makes one pass through the current working string making substitutions according to the rules added in the database.

2.14.14.5. LSYSTEM_DRAW (— id | — —)

Expects on stack: An L-system ID
Returns on stack: nothing
Operand Type: An L-System id
Description: Draws the LSYSTEM, by generating Gcodes

2.14.14.6. LSYSTEM_STRING (— id | — String)

Expects on stack: An L-system ID
Returns on stack: The current work string
Operand Type: An L-System id
Description: Returns the. Current working string as it exists at this point in time.