

MPCNC Forth Controller

Userguide

Version 0.5

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. Forth Constants and Variables

2.2.1.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 >
```

2.2.2. 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 >
```

2.2.3. @ (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

2.2.4. ! (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

2.2.5. ? (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.

2.3. 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

2.3.1. 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.

2.3.2. 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.

2.3.3. 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)

2.4. 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.

2.4.1. 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

2.4.2. 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

2.4.3. 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

2.5. Forth Comparators

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

2.5.1. = (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

2.5.2. < (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

2.5.3. > (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

2.5.4. <= (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

2.5.5. >= (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

2.6. Forth Zero Comparators

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

2.6.1. 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

2.6.2. 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

2.6.3. 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

2.7. 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.

2.8. Forth Looping Constructs

Two looping forms are currently supported by MPCNC 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.

2.8.1. 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
```

2.8.2. 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.

2.8.3. 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
```

2.8.4. 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>
```

2.9. Math Functions

2.9.1. 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

2.9.2. 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

2.9.3. 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

2.9.4. 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

2.9.5. 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.

2.9.6. 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

2.9.7. 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

2.9.8. 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.

2.9.9. 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.

2.9.10. 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

2.10. Miscellaneous Functions

2.10.1. 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.

2.10.2. (—Comment (— — | — —)

Expects on stack: Nothing

Returns on stack: Nothing

Operand Type: N/A

Description: Provides a way to comment your code
Comments start with “(“ and end with “)”

Example:

(this is a comment)

2.11. 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.

2.11.1. 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

2.11.2. 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

2.11.3. 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"

: $LEN ( string -- length ) \ "s-length"

: $. ( 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"

```

MPCNC Forth Controller User Guide

```
: $$>= ( 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"
```

2.12. GCODE Basic Extensions

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

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

2.12.1. 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

2.12.2. LASERLOFF (- - | - -)

Expects on stack: nothing

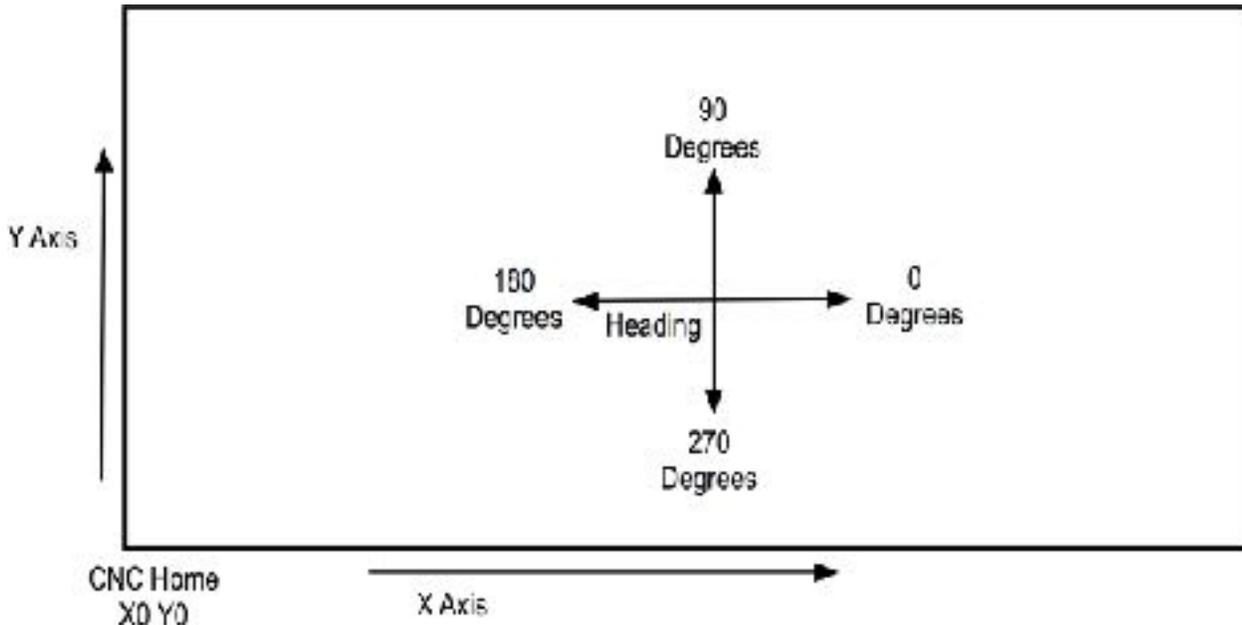
Returns on stack: nothing

Operand Type: N/A

Description: Sends power of zero using LASERCOMMANDPREFIX

2.13. Turtle Graphic Functions for the MPCNC

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



2.13.1. 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 focused

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 ROUTER, Raises Z by ROUTERSTANDOFFDISTANCE

```
if (zupwithmove) {
```

```
if (zupwithmove) {
```

2.13.2. 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.

2.13.3. 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.

2.13.4. 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.

2.13.5. 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)

2.13.6. 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).

2.13.7. 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.

2.13.8. ABSOLUTE (- - | - -)

Expects on stack: nothing

Returns on stack: nothing

Operand Type: N/A

Description: Sends a G90 command and waits for response.

2.13.9. RELATIVE (- - | - -)

Expects on stack: nothing

Returns on stack: nothing

Operand Type: N/A

Description: Sends a G91 command and waits for response.

2.13.10. 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

2.13.11. 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

2.13.12. 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.

GOXY
DRAWTO
CLEAR
FRAMEIT
CENTER
BOX
CUBE
CIRCLE
SIN
COS
TAN
DUPMOVE

