eForth Windows

Reference manual

version 1.0 - 9 novembre 2024
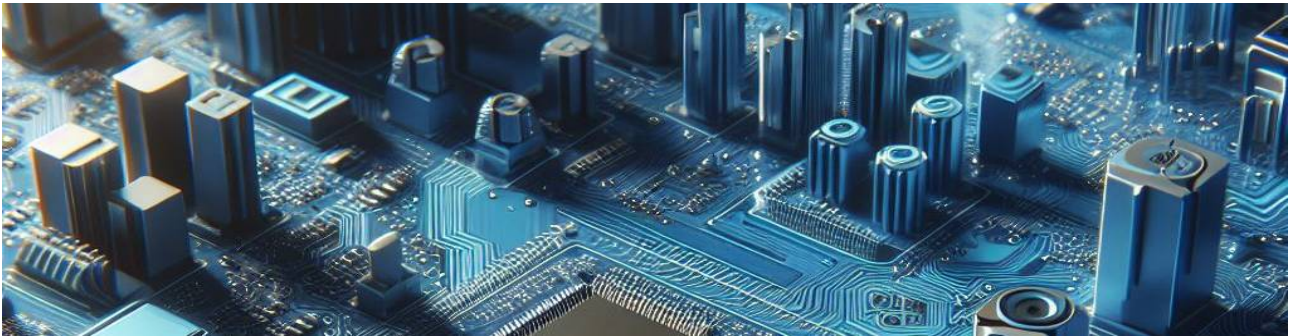
Author

- Marc PETREMANN   petremann@arduino-forth.com

Table des matières

forth

!   n addr --

Store n to address.

```
VARIABLE TEMPERATURE
32 TEMPERATURE !
```

d1 -- d2

Perform a division modulo the current numeric base and transform the rest of the division into a string of characters. The character is dropped in the buffer set to running <#

```
: hh ( c -- adr len)
    base @ >r  hex
    <# # # #>
    r> base !
  ;
 3 hh type  \ display 03
26 hh type  \ display 1a
```

#!   --

Behaves like \ for ESP32forth.

Serves as a text file header to indicate to the operating system (Unix-like) that this file is not a binary file but a script (set of commands). On the same line is specified the interpreter allowing to execute this script.

```
#! /usr/bin/env ueforth
```

#>   n -- addr len

Drop n. Make the pictured numeric output string available as a character string. *addr* and *len* specify the resulting character string.

```
\ display address in format: NNNN-NNNN
: DUMPaddr ( n -- )
    <# # # # #  [char] - hold # # # # #>
    type
  ;
```

#FS   r --

Converts a real number to a string. Used by f.

#s   n1 -- n=0

Converts the rest of n1 to a string in the character string initiated by <#.

```
: EUROS ( d1 --- str len)
   <#
   # #              \ convert € cents
   [char] , hold    \ add char "," to str buffer
   #s #>            \ convert rest after ","
  ;
15630. EUROS type   \ display 156,30 ok
```

#tib   -- n

Number of characters received in terminal input buffer.

'   exec: <space>name -- xt

Skip leading space delimiters. Parse name delimited by a space. Find name and return xt, the execution token for name.

When interpreting, ' xyz EXECUTE is equivalent to xyz.

'tib   -- addr

Pointer to Terminal Input Buffer.

(local)   a n --

Word used to manage the creation of local variables.

*   n1 n2 -- n3

Integer multiplication of two numbers.

```
6 3  *   \ push 18 operation 6*3
7 3  *   \ push 21 operation 7*3
-7 3 *   \ push -21
7 -3 *   \ push -21
-7 -3 *  \ push 21
```

*/   n1 n2 n3 -- n4

Multiply n1 by n2 producing the intermediate double-cell result d. Divide d by n3 giving the single-cell quotient n4.

```
5000 1000 4000 */ .    \ display    1250
```

*/MOD   n1 n2 n3 -- n4 n5

Multiply n1 by n2 producing the intermediate double-cell result d. Divide d by n3 producing the single-cell remainder n4 and the single-cell quotient n5.

```
50000 10 4001 */MOD .    \ display    124 3876
```

+   n1 n2 -- n3

Leave sum of n1 n2 on stack.

```
7 15 +     \ leave 22 on stack
```

+!   n addr --

Increments the contents of the memory address pointed to by addr.

```
variable valX
15 valX !
1 valX +!
valX ?    \ display 16
```

+loop   n --

Increment index loop with value n.

Mark the end of a loop `n1 0 do ... n2 +loop`.

```
: loopTest
   100 0 do
      i .
   5 +loop
  ;
loopTest  \ display 0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95
```

+to   n --- <valname>

add n to the content of *valname*

```
5 value FINAL-SCORE
1 +to FINAL-SCORE       \ increment content of FINAL-SCORE
FINAL-SCORE  .          \ display 6
```

,   x --

Append x to the current data section.

-   n1 n2 -- n1-n2

Subtract two integers.

```
6 3  - .  \ display 3
-6 3 - .  \ display -9
```

-rot   n1 n2 n3 -- n3 n1 n2

Inverse stack rotation. Same action than `rot rot`

.   n --

Remove the value at the top of the stack and display it as a signed single precision integer.

```
1 .                     \ display 1
1 2 .                   \ display 2  leave 1 on stack
1 2 + .                 \ display 3  addition 1 and 2, leave nothing on the
stack
6 3  * .                \ display 18
7 3  * 6 3 * + .        \ display 39 operation (7*3)+(6*3)
```

."   -- <string>

The word `."` can only be used in a compiled definition.

At runtime, it displays the text between this word and the delimiting **"** character end of string.

```
: TITLE
    ."      GENERAL MENU" CR
    ."      ============" ;
: line1
    ." 1.. Enter datas" ;
: line2
    ." 2.. Display datas" ;
: last-line
    ." F.. end program" ;
: MENU ( ---)
    title cr cr cr
    line1 cr cr
    line2 cr cr
    last-line ;
```

.s   --

Displays the content of the data stack, with no action on the content of this stack.

/   n1 n2 -- n3

Divide n1 by n2, giving the single-cell quotient n3.

```
6 3  / .  \ display 2 opération 6/3
7 3  / .  \ display 2 opération 7/3
8 3  / .  \ display 2 opération 8/3
9 3  / .  \ display 3 opération 9/3
```

/mod   n1 n2 -- n3 n4

Divide n1 by n2, giving the single-cell remainder n3 and the single-cell quotient n4.

```
22 7 /MOD . .     \ display   3 1
```

0<   x1 --- fl

Test if x1 is less than zero.

0<>   n -- fl

Leave -1 if n <> 0

0=   x -- fl

flag is true if and only if x is equal to zero.

```
5 0=       \ push  FALSE on stack
0 0=       \ push  TRUE  on stack
```

1+   n -- n+1

Increments the value at the top of the stack.

1-   n -- n-1

Decrements the value at the top of the stack.

1/F   r -- r'

Performs a 1/r operation.

```
12e 1/F f.  \ display 0.083333  (op: 1/12)
```

2!   d addr --

Store double precision value in memory address addr.

2*   n -- n*2

Multiply n by two.

2/   n -- n/2

Divide n by two.

n/2 is the result of shifting n one bit toward the least-significant bit, leaving the most-significant bit unchanged

```
24 2/ .     \ display   12
25 2/ .     \ display   12
26 2/ .     \ display   13
```

2@   addr -- d

Leave on stack double precision value d stored at address addr.

2drop   n1 n2 n3 n4 -- n1 n2

Removes the double-precision value from the top of the data stack.

```
1 2 3 4 2drop   \ leave 1 2 on top of stack
```

2dup   n1 n2 -- n1 n2 n1 n2

Duplicates the double precision value n1 n2.

```
1 2 2dup  \ leave 1 2 1 2 on stack
```

4*   n -- n*4

Multiply n by four.

4/   n -- n/4

Divide n by four.

:   comp: -- <word> | exec: --

Skip leading space delimiters. Parse name delimited by a space. Create a definition for name, called a "colon definition". Enter compilation state and start the current definition.

Subsequent execution of NOM performs the execution sequence words compiled in his "colon" definition.

After : NOM, the interpreter enters compile mode. All non-immediate words are compiled in the definition, the numbers are compiled in literal form. Only immediate words or placed in square brackets (words [and]) are executed during compilation to help control it.

A "colon" definition remains invalid, ie not inscribed in the current vocabulary, as long as the interpreter did not execute ; (semi-colon).

```
: NAME  nomex1 nomex2 ... nomexn ;
NAME   \ execute NAME
```

:noname   -- cfa-addr

Define headerless forth code. cfa-addr is the code execution of a definition.

```
:noname s" Saturday" ;
:noname s" Friday" ;
:noname s" Thursday" ;
:noname s" Wednesday" ;
:noname s" Tuesday" ;
:noname s" Monday" ;
```

```
:noname s" Sunday" ;

create (ENday) ( --- addr)
        , , , , , , ,

:noname s" Samedi" ;
:noname s" Vendredi" ;
:noname s" Jeudi" ;
:noname s" Mercredi" ;
:noname s" Mardi" ;
:noname s" Lundi" ;
:noname s" Dimanche" ;

create (FRday) ( --- addr)
        , , , , , , ,

defer (day)

: ENdays
    ['] (ENday) is (day) ;

: FRdays
    ['] (FRday) is (day) ;

3 value dayLength
: .day
    (day)
    swap cell *
    + @ execute
    dayLength ?dup if
        min
    then
    type
;
ENdays
0 .day \ display Sun
1 .day \ display Mon
2 .day \ display Tue
FRdays  ok
0 .day \ display Dim
1 .day \ display Lun
2 .day \ display Mar
```

; --

Immediate execution word usually ending the compilation of a "colon" definition.

```
: NAME
    nomex1 nomex2
    nomexn ;
```

<   n1 n2 -- fl

Leave fl true if n1 < n2

```
4 10 <=   \ leave -1 on stack
4 4  <=   \ leave  0 on stack
4 3  <=   \ leave  0 on stack
```

<#   n --

Marks the start of converting a integer number to a string of characters.

```
\ display address in format: NNNN-NNNN
: DUMPaddr ( n -- )
   <# # # # #  [char] - hold # # # # #>
   type
 ;

\ display byte in format: NN
: DUMPbyte ( c -- )
   <# # # #>
   type
 ;
```

<=   n1 n2 -- fl

Leave fl true if n1 <= n2

```
4 10 <=    \ leave -1 on stack
4 4  <=    \ leave -1 on stack
4 3  <=    \ leave  0 on stack
```

<>   x1 x2 -- fl

flag is true if and only if x1 is different x2.

```
5 5 <>        \ push  FALSE on stack
5 4 <>        \ push  TRUE  on stack
```

=   n1 n2 -- fl

Leave fl true if n1 = n2

```
4 10 =    \ leave  0 on stack
4 4  =    \ leave -1 on stack
```

>   x1 x2 -- fl

Test if x1 is greater than x2.

>=   x1 x2 -- fl

flag is true if and only if x1 is equal x2.

```
5 5 >=        \ push  FALSE on stack
5 4 >=        \ push  TRUE  on stack
```

>body   cfa -- pfa

pfa is the data-field address corresponding to cfa.

>flags   xt -- flags

Convert cfa address to flags address.

>in   -- addr

Number of characters consumed from TIB

```
tib >in @ type
\ display:
tib >in @
```

>link   cfa -- cfa2

Converts the cfa address of the current word into the cfa address of the word previously defined in the dictionary.

```
' dup >link    \ get cfa from word defined before dup
>name type     \ display "XOR"
```

>link&   cfa -- lfa

Transforms the execution address of the current word into the link address of this word. This link address points to the cfa of the word defined before this word.

Used by `>link`

>name   cfa -- nfa len

finds the name field address of a token from its code field address.

```
' words        \ push cfa of 'words' on stack
>name          \ convert cfa of 'words' in nfa field follewed by len
type           \ display 'words'
```

>name-length   cfa -- n

Transforms a cfa address into the length of the word name of this cfa address. Word used by `vlist`

>r   S: n -- R: n

Transfers n to the return stack.

This operation must always be balanced with `r>`

```
\ display n in binary format
: b. ( n -- )
    base @ >r
    binary .
    r> base !
  ;
```

?   addr -- c

Displays the content of any variable or address.

?do   n1 n2 --

Executes a `do loop` or `do +loop` loop if n1 is strictly greater than n2.

```
DECIMAL
: qd ?DO I LOOP ;
   789    789 qd \
 -9876 -9876 qd \
     5      0 qd \  display: 0 1 2 3 4
```

?dup   n -- n | n n

Duplicate n if n is not nul.

@   addr -- n

Retrieves the integer value n stored at address addr.

```
TEMPERATURE @
```

abort   --

Raises an exception and interrupts the execution of the word and returns control to the interpreter.

abort"   comp: --

Displays an error message and aborts any FORTH execution in progress.

```
: abort-test
   if
       abort" stop program"
   then
   ." continue program"
 ;

0 abort-test   \ display: continue program
1 abort-test   \ display: stop program ERROR
```

abs   n -- n'

Return the absolute value of n.

```
-7 abs .    \ display 7
```

accept   addr n -- n

Accepts n characters from the keyboard (serial port) and stores them in the memory area pointed to by addr.

```
create myBuffer 100 allot
myBuffer 100 accept     \ on prompt, enter: This is an example
myBuffer swap type      \ display: This is an example
```

afliteral   r:r --

Compiles a real number. Used by `fliteral`

aft   --

Jump to THEN in FOR-AFT-THEN-NEXT loop 1st time through.

```
: test-aft1 ( n -- )
  FOR
    ."  for "      \ first iteration
    AFT
      ."  aft "    \ following iterations
    THEN
    I .            \ all iterations
  NEXT ;
3 test-aft1
\ display for 3 aft 2 aft 1 aft 0
```

again   --

Mark the end on an infinit loop of type `begin ... again`

```
: test ( -- )
   begin
      ." Diamonds are forever" cr
   again
 ;
```

align   --

Align the current data section dictionary pointer to cell boundary.

aligned   addr1 -- addr2

addr2 is the first aligned address greater than or equal to addr1.

allot   n --

Reserve n address units of data space.

also   --

Duplicate the vocabulary at the top of the vocabulary stack.

analogRead   pin -- n

Analog read from 0-4095.

Use to read analog value. `analogRead` has only one argument which is a pin number of the analog channel you want to use.

```
\ solar cell connected on pin G34
34 constant SOLAR_CELL

: init-solar-cell ( -- )
    SOLAR_CELL input pinMode
  ;

: solar-cell-read ( -- n )
    SOLAR_CELL analogRead
  ;
```

AND   n1 n2 --- n3

Execute logic AND.

The words `AND`, `OR`, and `XOR` perform operations binary **bitwise** logic on single-precision integers at the top of the data stack.

```
0   0 and .      \ display 0
0  -1  and .     \ display 0
-1   0 and .     \ display 0
-1   -1  and .  \ display -1
```

ansi   --

Selects the `ansi` vocabulary.

argc   -- n

Push content of `'argc` on stack

ARSHIFT   x1 u -- x2

Arithmetic right shift of u

asm   --

Select the `asm` vocabulary.

assembler --DELETE   --

Alias for `asm`.

Select the `asm` vocabulary.

assert   fl --

For tests and asserts.

at-xy   x y --

Positions the cursor at the x y coordinates.

```
: menu ( -- )
   page
   10 4 at-xy
      0 bg 7 fg   ."  Your choice, press: " normal
   12 5 at-xy  ." A - accept"
   12 6 at-xy  ." D - deny"
  ;
```

base   -- addr

Single precision variable determining the current numerical base.

The **BASE** variable contains the value 10 (decimal) when FORTH starts.

```
DECIMAL      \ select decimal base
2 BASE !     \ selevt binary base

\ other example
: GN2 \ ( -- 16 10 )
   BASE @ >R HEX BASE @ DECIMAL BASE @ R> BASE !
   ;
```

begin   --

Mark start of a structure **begin..until**, **begin..again** or **begin..while..repeat**

```
: endless ( -- )
   0
   begin
       dup . 1+
   again
  ;
```

bg   color[0..255] --

Selects the background display color. The color is in the range 0..255 in decimal.

```
: testBG ( -- )
  normal
  256 0 do
    i bg ." X"
  loop ;
```

BIN   mode -- mode'

Modify a file-access method to include BINARY.

BINARY   --

Select binary base.

```
255 BINARY .    \ display 11111111
DECIMAL         \ return to decimal base
```

bl   -- 32

Value 32 on stack.

```
\ definition of bl
: bl  ( -- 32 )
    32
  ;
```

blank   addr len --

If len is greater than zero, store byte $20 (space) in each of len consecutive characters of memory beginning at addr.

block   n -- addr

Get addr 1024 byte for block n.

block-fid   -- n

Flag indicating the state of a block file.

block-id   -- n

Pointer to a block file.

buffer   n - addr

Get a 1024 byte block without regard to old contents.

bye   --

Word defined by `defer`.

c!   c addr --

Stores an 8-bit c value at address addr.

```
36 constant DDRB  \ data direction register for PORT B on Arduino
32 DDRB c!        \ same as 35 32 c!
```

c,   c --

Append c to the current data section.

```
create myDatas
    36 c,   42 c,  24 c,  12 c,
myDatas 1+ c@   \ push 42 on stack
```

c@   addr -- c

Retrieves the 8-bit c value stored at address addr.

```
35 constant PINB  \ adresse registre données PIN de PORT B sur Arduino
PINB c@           \ empile contenu registre pointé par PINB
```

camera-server   --

Select `camera-server` vocabulary.

CASE   --

```
: day ( n -- addr len )
    CASE
        0 OF s" Sunday"      ENDOF
        1 OF s" Monday"      ENDOF
        2 OF s" Tuesday"     ENDOF
        3 OF s" Wednesday"   ENDOF
        4 OF s" Thursday"    ENDOF
        5 OF s" Friday"      ENDOF
        6 OF s" Saturday"    ENDOF
    ENDCASE
  ;
```

cat   -- <path>

Display the file content.

```
cat /tools/dumpTool.txt
\ display content of file dumpTool.txt
\ if this file was edited and saved in /spiffs/ file system
```

catch   cfa -- fl

Initializes an action to perform in the event of an exception triggered by `throw`.

cell   -- 8

Return number of bytes in a 64-bit integer.

```
cell .    \ display 8
```

cell+   n -- n'

Increment `CELL` content.

cell/   n -- n'

Divide **CELL** content.

cells   n -- n'

Multiply **CELL** content.

Allows you to position yourself in an array of integers.

```
create table ( -- addr)
    1 ,  5 ,  10 ,  50 ,  100 , 500 ,
\ get values indexed 0 and 3 from table
table 0 cells + @ .   \ display 1
table 3 cells + @ .   \ display 50
```

char   -- <string>

Word used in interpretation only.

Leave the first character of the string following this word.

```
char v .          \ display: 118 (ascii code for "v")
char house .      \ display: 104 - code for "h"
```

CLOSE-FILE   fileid -- ior

Close an open file.

cmove   c-addr1 c-addr2 u --

If u is greater than zero, copy u consecutive characters from the data space starting at c-addr1 to that starting at c-addr2, proceeding character-by-character from lower addresses to higher addresses.

code   -- <:name>

Defines a word whose definition is written in assembly language.

```
code my2*
  a1 32 ENTRY,
  a8 a2 0 L32I.N,
  a8 a8 1 SLLI,
  a8 a2 0 S32I.N,
  RETW.N,
end-code
```

constant   comp: n -- <name> | exec: -- n

Define a constant.

```
$00000001 constant SDL_INIT_TIMER            \ timer subsystem
$00000010 constant SDL_INIT_AUDIO            \ audio subsystem
```

```
$00000020 constant SDL_INIT_VIDEO              \ video subsystem; automatically
initializes the events subsystem
$00000200 constant SDL_INIT_JOYSTICK           \ joystick subsystem; automatically
initializes the events subsystem
```

context   -- addr

Pointer to pointer to last word of context vocabulary

copy   from to --

Copy contents of block 'from' to block 'to'

cp   -- "src" "dst"

Copy "src" file to "dst".

cr   --

Show a new line return.

```
: .result ( ---)
    ." Port analys result" cr
    . "pool detectors" cr ;
```

CREATE   comp: -- <name> | exec: -- addr

The word CREATE can be used alone.

The word after CREATE is created in the dictionary, here DATAS. The execution of the word
thus created deposits on the data stack the memory address of the parameter zone. In
this example, we have compiled 4 8-bit values. To recover them, it will be necessary to
increment the address stacked with the value shifting the data to be recovered.

```
\ Peripherals accessed by the CPU via 0x3FF40000 ~ 0x3FF7FFFF address space
\ (DPORT address) can also be accessed via 0x60000000 ~ 0x6003FFFF
\ (AHB address). (0x3FF40000 + n) address and (0x60000000 + n)
\ address access the same content, where n = 0 ~ 0x3FFFF.
create uartAhbBase
    $60000000 ,
    $60010000 ,
    $6002E000 ,

: REG_UART_AHB_BASE { idx -- addr }      \ id=[0,1,2]
    uartAhbBase  idx cell *  + @
  ;
```

CREATE-FILE   a n mode -- fh ior

Create a file on disk, returning a 0 ior for success and a file id.

current   -- cfa

Pointer to pointer to last word of current vocabulary

```
: test ( -- )
   ." only for test" ;
current @ @ >name type   \ display test
```

DECIMAL   --

Selects the decimal number base. It is the default digital base when FORTH starts.

```
HEX
FF DECIMAL .   \ display 255
```

default-key   -- c

Execute `win-key`.

default-key?   -- fl

Execute `win-key?`.

default-type   addr len --

Execute `win-type`.

defer   -- <vec-name>

Define a deferred execution vector.

`vec-name` execute the word whose execution token is stored in vec-name's data space.

DEFINED?   -- <word>

Returns a non-zero value if the word is defined. Otherwise returns 0.

```
\ other example:
DEFINED? --DAout [if] forget --DAout  [then]
create --DAout
```

definitions   --

Make the compilation word list the same as the first word list in the search order. Specifies that the names of subsequent definitions will be placed in the compilation word list. Subsequent changes in the search order will not affect the compilation word list.

```
VOCABULARY LOGO          \ create vocabulary LOGO
LOGO DEFINITIONS         \ will set LOGO context vocabulary
: EFFACE
   page ;                \ create word EFFACE in LOGO vocabulary
```

depth   -- n

n is the number of single-cell values contained in the data stack before n was placed on the stack.

```
\ test this after reset:
depth         \ leave 0 on stack
10 32 25
depth         \ leave 3 on stack
```

digitalWrite   pin value --

Set GPIO pin state.

```
17 constant TRIGGER_ON       \ green LED
16 constant TRIGGER_OFF      \   red LED

: init-trigger-state ( -- )
    TRIGGER_ON  output pinMode
    TRIGGER_OFF output pinMode
  ;

TRIGGER_ON HIGH digitalWrite
```

do   n1 n2 --

Set up loop control parameters with index n2 and limit n1.

```
: testLoop
    256 32 do
        I emit
    loop
  ;
```

DOES>   comp: -- | exec: -- addr

The word CREATE can be used in a new word creation word...

Associated with DOES>, we can define words that say how a word is created then executed.

drop   n --

Removes the single-precision integer that was there from the top of the data stack.

```
2 5 8  drop   \ leave 2 and 5 on stack
```

dump   a n --

Dump a memory region

dump-file   addr len addr2 len2 --

Transfers the contents of a text string addr len to a file pointed by addr2 len2

The content of the */spiffs/autoexec.fs* file is automatically interpreted and/or compiled when ESP32Forth starts.

This feature can be leveraged to set up WiFi access when starting ESP32Forth by injecting the access parameters like this:

```
r| z" NETWORK-NAME" z" PASSWORD" webui |
s" /spiffs/autoexec.fs"
dump-file
```

dup   n -- n n

Duplicates the single-precision integer at the top of the data stack.

```
: SQUARE ( n --- nE2)
    DUP * ;
5 SQUARE .     \ display 25
10 SQUARE .    \ display 100
```

echo   -- addr

Variable. Value is -1 by default. If 0, commands are not displayed.

editor   --

Select `editor`.

- `l` lists the content of the current block

- `n` select the next block

- `p` select the previous block

- `wipe` empties the content of the current block

- `d` delete line n. The line number must be in the range 0..14. The following lines go up.

  Example: `3 D` erases the content of line 3 and brings up the content of lines 4 to 15.

- `e` erases the content of line n. The line number must be in the range 0..15. The other lines do not go up.

- `a` inserts a line n. The line number must be in the range 0..14. The lines located after the inserted line come down.

  Example: `3 A test` inserts **test** on line 3 and move the contents of lines 4 to 15.

- **r** replaces the content of line n. Example: `3 R test` replace the contents of line 3 with **test**

else   --

Word of immediate execution and used in compilation only. Mark a alternative in a control structure of the type `IF ... ELSE ... THEN`

At runtime, if the condition on the stack before `IF` is false, there is a break in sequence with a jump following `ELSE`, then resumed in sequence after `THEN`.

```
: TEST ( ---)
   CR ." Press a key " KEY
   DUP 65 122 BETWEEN
   IF
      CR 3 SPACES ." is a letter "
   ELSE
      DUP 48 57 BETWEEN
      IF
         CR 3 SPACES ." is a digit "
      ELSE
         CR 3 SPACES ." is a special character "
      THEN
   THEN
   DROP ;
```

emit   x --

If x is a graphic character in the implementation-defined character set, display x.

The effect of `EMIT` for all other values of x is implementation-defined.

When passed a character whose character-defining bits have a value between hex 20 and 7E inclusive, the corresponding standard character is displayed. Because different output devices can respond differently to control characters, programs that use control characters to perform specific functions have an environmental dependency. Each `EMIT` deals with only one character.

```
65 emit     \ display A
66 emit     \ display B
```

empty-buffers   --

Empty all buffers.

ENDCASE   --

Marks the end of a `CASE OF ENDOF ENDCASE` structure

```
: day ( n -- addr len )
   CASE
      0 OF s" Sunday"     ENDOF
```

```
         1 OF s" Monday"      ENDOF
         2 OF s" Tuesday"     ENDOF
         3 OF s" Wednesday"   ENDOF
         4 OF s" Thursday"    ENDOF
         5 OF s" Friday"      ENDOF
         6 OF s" Saturday"    ENDOF
      ENDCASE
   ;
```

ENDOF   --

Marks the end of a `OF .. ENDOF` choice in the control structure between `CASE ENDCASE`.

```
: day ( n -- addr len )
   CASE
         0 OF s" Sunday"      ENDOF
         1 OF s" Monday"      ENDOF
         2 OF s" Tuesday"     ENDOF
         3 OF s" Wednesday"   ENDOF
         4 OF s" Thursday"    ENDOF
         5 OF s" Friday"      ENDOF
         6 OF s" Saturday"    ENDOF
      ENDCASE
   ;
```

erase   addr len --

If len is greater than zero, store byte $00 in each of len consecutive characters of memory beginning at addr.

evaluate   addr len --

Evaluate the content of a string.

```
s" words"
evaluate    \ execute the content of the string, here: words
```

EXECUTE   xt --

Execute word at xt.

Take the execution address from the data stack and executes that token. This powerful word allows you to execute any token which is not a part of a token list.

exit   --

Aborts the execution of a word and gives back to the calling word.

Typical use: `: X ... test IF ... EXIT THEN ... ;`

At run time, the word `EXIT` will have the same effect as the word `;`

extract   n base -- n c

Extract the least significant digit of n. Leave on the stack the quotient of n/base and the ASCII character of this digit.

F*   r1 r2 -- r3

Multiplication of two real numbers.

```
1.35e 2.2e F*
F.  \ display 2.969999
```

F**   r_val r_exp -- r

Raises a real r_val to the power r_exp.

```
2e 3e f** f.     \ display 8.000000
  2e 4e f** f.     \ display 16.000000
 10e 1.5e f** f.  \ display 31.622776
```

F+   r1 r2 -- r3

Addition of two real numbers.

```
3.75e 5.21e F+
F.  \ display 8.960000
```

F-   r1 r2 -- r3

Subtraction of two real numbers.

```
10.02e 5.35e F-
F.  \ display 4.670000
```

f.   r --

Displays a real number. The real number must come from the real stack.

```
pi f.    \ display 3.141592
```

f.s   --

Display content of reals stack.

```
2.35e
36.512e
f.s  \ display: <2> 2.350000 36.511996
```

F/   r1 r2 -- r3

Division of two real numbers.

```
22e 7e F/    \ PI approximation
F.           \ display 3.142857
```

F0<   r -- fl

Tests if a real number is less than zero.

```
5e F0<      \ leave  0 on stack
-3e F0<     \ leave -1 on stack
```

F0=   r -- fl

Indicates true if the real is null.

```
3e 3e F- F0= .   \ display -1
```

f<   r1 r2 -- fl

fl is true if r1 < r2

```
3.2e 5.25e f<
.  \ display -1
```

f<=   r1 r2 -- fl

fl is true if r1 <= r2.

```
3.2e 5.25e f<=
.  \ display -1
5.25e 5.25e f<=
.  \ display -1
8.3e 5.25e f<=
.  \ display 0
```

f<>   r1 r2 -- fl

fl is true if r1 <> r2.

```
3.2e 5.25e f<>
.  \ display -1
5.25e 5.25e f<>
.  \ display 0
```

f=   r1 r2 -- fl

fl is true if r1 = r2.

```
3.2e 5.25e f=
.  \ display 0
5.25e 5.25e f=
.  \ display -1
```

f> r1 r2 -- fl

fl is true if r1 > r2.

```
3.2e 5.25e f>
.   \ display 0
```

f>= r1 r2 -- fl

fl is true if r1 > = r2.

```
3.2e 5.25e f>=
.   \ display 0
5.25e 5.25e f>=
.   \ display -1
8.3e 5.25e f>=
.   \ display -1
```

F>S r -- n

Convert a real to an integer. Leaves the integer part on the data stack if the real has fractional parts.

```
3.5e F>S .   \ display 3
```

FABS r1 -- r1'

Returns the absolute value of a real number.

```
-2e FABS F.  \ display 2.000000
```

FATAN2 r-tan -- r-rad

Calculates the angle in radians from the tangent.

```
0.5e fatan2 f.    \ display 1.325917
  1e fatan2 f.    \ display 0.785398
```

fconstant comp: r -- <name> | exec: -- r

Defines a constant of type real.

```
9.80665e fconstant g    \ gravitation constant on Earth
g f.  \ display 9.806649
```

FCOS r1 -- r2

Calculates the cosine of an angle expressed in radians.

```
pi 2e f/       \ calc angle 90 deg
FCOS F.        \ display 0.000000
```

fdepth   -- n

n is the number of reals values contained in the real stack.

FDROP   r1 --

Drop real r1 from real stack.

FDUP   r1 -- r1 r1

Duplicate real r1 from real stack.

FEXP   ln-r -- r

Calculate the real corresponding to e EXP r

```
4.605170e FEXP F.    \ display 100.000018
```

fg   color[0..255] --

Selects the text display color. The color is in the range 0..255 in decimal.

```
: testFG ( -- )
  256 0 do
    i fg ." X"
  loop ;
```

file-exists?   addr len --

Tests if a file exists. The file is designated by a character string.

```
s" /spiffs/dumpTool.txt" file-exists?
```

FILE-POSITION   fileid -- ud ior

Return file position, and return ior=0 on success.

FILE-SIZE   fileid -- ud ior

Get size in bytes of an open file as a double number, and return ior=0 on success.

fill   addr len c --

If len is greater than zero, store c in each of len consecutive characters of memory beginning at addr.

FIND   addr len -- xt | 0

Find a word in dictionnary.

```
32 string t$
s" vlist" t$ $!
```

```
t$ find     \ push cfa of VLIST on stack
```

fliteral   r:r --

Immediate execution word. Compiles a real number.

FLN   r -- ln-r

Calculates the natural logarithm of a real number.

```
100e FLN f.     \ display 4.605170
```

FLOOR   r1 -- r2

Rounds a real down to the integer value.

```
45.67e FLOOR F.     \ display 45.000000
```

flush   --

Save and empty all buffers.

After editing the contents of a block file, running `flush` ensures that changes to the contents of blocks are saved.

FLUSH-FILE   fileid –- ior

Attempt to force any buffered information written to the file referred to by fileid to be written to mass storage. If the operation is successful, ior is zero.

FMAX   r1 r2 -- r1|r2

Let the greatest real of r1 or r2.

```
3e 4e FMAX F.     \ display 4.000000
```

FMIN   r1 r2 -- r1|r2

Let the smaller real of r1 or r2.

```
3e 4e FMIN F.     \ display 3.000000
```

FNEGATE   r1 -- r1'

Reverses the sign of a real number.

```
5e FNEGATE f.      \ display -5.000000
-7e FNEGATE f.      \ display  7.000000
```

FNIP   r1 r2 -- r2

Delete second element on reals stack.

```
2.5e 4.32e
fnip
f.s  \ display: <1> 4.320000
```

for   n --

Marks the start of a loop `for .. next`

WARNING: the loop index will be processed in the interval [n..0], i.e. n+1 iterations, which is contrary to the other versions of the FORTH language implementing FOR..NEXT (FlashForth).

```
: myLoop ( ---)
   10 for
       r@ . cr  \ display loop index
   next
 ;
```

forget   -- <name>

Searches the dictionary for a name following it. If it is a valid word, trim dictionary below this word. Display an error message if it is not a valid word.

forth   --

Select the `FORTH` vocabulary in the word search order to execute or compile words.

forth-builtins   -- cfa

Entry point of `forth` vocabulary.

FOVER   r1 r2 -- r1 r2 r1

Duplicate second real on reals stack.

```
2.6e 3.4e fover
f.s  \ display <3> 2.600000 3.400000 2.600000
```

fp0   -- addr

Points to the bottom of reals stack.

FP@   -- addr

Retrieves the stack pointer address of the reals.

freq   chan freq --

sets frequency freq n to channel chan.

Use `ledcWriteTone`

FSIN   r1 -- r2

Calculates the sine of an angle expressed in radians.

```
pi 2e f/       \ calc angle 90¨ deg
FSIN F.        \ display 1.000000
```

FSINCOS   r1 -- rcos rsin

Calculates the cosine eand sine of an angle expressed in radians.

```
pi 4e f/
FSINCOS f. f.   \ display 0.707106 0.707106
pi 2e f/
FSINCOS f. f.   \ display 0.000000 1.000000
```

fsqrt   r1 -- r2

Square root of a real number.

```
64e fsqrt
F.            \ display 8.000000
```

FSWAP   r1 r2 -- r1 r2

Reverses the order of the two values on the ESP32Forth real stack.

```
3.75e 5.21e FSWAP
F.   \ display 3.750000
F.   \ display 5.210000
```

fvariable   comp: -- <name> | exec: -- addr

Defines a floating point variable.

```
fvariable arc
pi 0.5e F*  \ angle 90° in radian   --  PI/2
arc SF!
arc SF@ f.  \ display 1.570796
```

graphics   --

select `graphics` vocabulary.

handler   -- addr

Ticket for interruptions.

here   -- addr

Leave the current data section dictionary pointer.

The dictionary pointer is incremented as the words are compiled and variables and data tables are defined.

```
here u.         \ display 1073709120
: null ;
here u.         \ display 1073709144
```

HEX   --

Selects the hexadecimal digital base.

```
255 HEX .    \ display FF
DECIMAL      \ return to decimal base
```

hld   -- addr

Pointer to text buffer for number output.

hold   c --

Inserts the ASCII code of an ASCII character into the character string initiated by <#.

i   -- n

n is a copy of the current loop index.

```
: mySingleLoop ( -- )
    cr
    10 0 do
        i .
    loop
  ;
mySingleLoop
\ display 0 1 2 3 4 5 6 7 8 9
```

if   fl --

The word IF is executed immediately.

IF marks the start of a control structure for type IF..THEN or IF..ELSE..THEN.

```
: WEATHER? ( fl ---)
    IF
        ." Nice weather "
    ELSE
        ." Bad weather "
```

```
    THEN ;
1 WEATHER?      \ display: Nice weather
0 WEATHER?      \ display: Bad weather
```

immediate   --

Make the most recent definition an immediate word.

Sets the compile-only lexicon bit in the name field of the new word just compiled. When the interpreter encounters a word with this bit set, it will not execute this word, but spit out an error message. This bit prevents structure words to be executed accidentally outside of a compound word.

include   -- <:name>

Loads the contents of a file designated by <name>.

The word `include` can only be used from the terminal.

To load the contents of a file from another file, use the word `included`.

```
include /spiffs/dumpTool.txt
\ load content of dump.txt

\ to include a file from an other file, use included
s" /spiffs/dumpTool.txt" included
```

included   addr len --

Loads the contents of a file from the SPIFFS filesystem, designated by a character string.

The word `included` can be used in a FORTH listing stored in the SPIFFS file system.

For this reason, the filename to load should always be preceded by *spiffs/*

```
s" /spiffs/dumpTool.txt" included
```

included?   addr len -- f

Tests whether the file named in the character string has already been compiled.

internalized   --

select `internalized` vocabulary.

internals   --

Select `internals` vocabulary.

invert   x1 -- x2

Complement to one of x1. Acts on 16 or 32 bits depending on the FORTH versions.

```
1 invert .   \ display -2
```

is **--**

Affecte le code d'exécution d'un mot à un mot d'exécution vectorisée.

j **-- n**

n is a copy of the next-outer loop index.

```
: myDoubleLoop ( -- )
    cr
    10 0 do
        cr
        10 0 do
            i 1+  j 1+  * .
        loop
    loop
  ;
myDoubleLoop
\ display:
1 2 3 4 5 6 7 8 9 10
2 4 6 8 10 12 14 16 18 20
3 6 9 12 15 18 21 24 27 30
4 8 12 16 20 24 28 32 36 40
5 10 15 20 25 30 35 40 45 50
6 12 18 24 30 36 42 48 54 60
7 14 21 28 35 42 49 56 63 70
8 16 24 32 40 48 56 64 72 80
9 18 27 36 45 54 63 72 81 90
10 20 30 40 50 60 70 80 90 100
```

k **-- n**

n is a copy of the next-next-outer loop index.

```
: myTripleLoop ( -- )
    cr
    5 0 do
        cr
        5 0 do
            cr
            5 0 do
                i 1+  j 1+  k 1+  * * .
            loop
        loop
    loop
  ;
myTripleLoop
```

key **-- char**

Waits for a key to be pressed. Pressing a key returns its ASCII code.

```
key .    \ display 97 if key "a" is active
key .    \ affiche 65 if key "A" is active
```

key?   -- fl

Returns *true* if a key is pressed.

```
: keyLoop
   begin
   key? until
 ;
```

L!   n addr --

Store a value n.

L,   n --

Word not implemented in eForth Windows.

Stores a value in 32-bit format in the dictionary.

Definition:

```
DEFINED? L, invert [IF]
\ compile 32 bits value in dictionnary
: L,  ( u -- )
   dup c,
   8 rshift dup c,
   8 rshift dup c,
   8 rshift dup c,
   drop
  ;
[THEN]
```

latestxt   -- xt

Stacks the execution code (cfa) address of the last compiled word.

```
: txtxtx  ;
latest
>name type  \ display txtxtx
```

leave   --

Prematurely terminates the action of a `do..loop` loop.

LED   -- 2

Pin 2 value for LED on the board. Does not work with all cards.

list   n --

Displays the contents of block n.

literal   x --

Compiles the value x as a literal value.

```
: valueReg ( --- n)
   [ 36 2 * ] literal ;

\ equivalent to:
: valueReg ( --- n)
   72 ;
```

load   n --

Evaluate a block.

load preceded by the number of the block you want to execute and/or compile the content. To compile the content of our block 0, we will execute **0 load**

loop   --

Add one to the loop index. If the loop index is then equal to the loop limit, discard the loop parameters and continue execution immediately following the loop. Otherwise continue execution at the beginning of the loop.

```
: myLoop
  128 32 do
    i emit
  loop ;
```

ls   -- "path"

Displays the contents of a file path.

```
ls /spiffs/  \ display:
dump.txt
```

LSHIFT   x1 u -- x2

Shift to the left of u bits by the value x1.

```
8 2 lshift .  \ display 32
```

max   n1 n2 -- n1|n2

Leave the unsigned larger of u1 and u2.

MDNS.begin   name-z -- fl

Start multicast dns.

```
z" forth" MDNS.begin
```

min   **n1 n2 -- n1|n2**

Leave min of n1 and n2

mod   **n1 n2 -- n3**

Divide n1 by n2, giving the single-cell remainder n3.

The modulo function can be used to determine the divisibility of one number by another.

```
21 7 mod . \ display 0
22 7 mod . \ display 1
23 7 mod . \ display 2
24 7 mod . \ display 3

: DIV? ( n1 n2 ---)
    OVER OVER MOD CR
    IF
        SWAP . ." is not "
    ELSE
        SWAP . ." is "
    THEN
    ." divisible by " .
 ;
```

ms   **n --**

Waiting in millisencondes.

For long waits, set a wait word in seconds.

```
500 ms \ delay for 1/2 second

: seconds ( n --)
    0
    for
        1000 ms
    next
  ;
12 seconds \ delay for 12 seconds
```

MS-TICKS   **-- n**

System ticks. One tick per millisecond.

Useful for measuring the execution time of a definition.

mv   **-- "src" "dest"**

Rename "src" file to "dst".

n.   **n --**

Display anay value n in decimal format.

negate   **n -- -n'**

Two's complement of n.

```
5 negate .    \ display -5
```

next   **--**

Marks the end of a loop **for .. next**

```
: myLoop
  24 for
    r@ .
  next ;
myLoop   \ display: 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3
2 1 0
```

nip   **n1 n2 -- n2**

Remove n1 from the stack.

nl   **-- 10**

Value 10 on stack.

normal   **--**

Disables selected colors for display.

NULL   **-- 0**

Word not implemented in eForth Windows.

Definition:

```
DEFINED? NULL invert [IF]
0 constant NULL
[THEN]
```

OCTAL   **--**

Selects the octal digital base.

```
255 OCTAL .      \ display 377
DECIMAL          \ return to decimal base
```

OF   **n --**

Marks a **OF .. ENDOF** choice in the control structure between **CASE ENDCASE**

If the tested value is equal to the one preceding **OF**, the part of code located between **OF ENDOF** will be executed.

```
: day ( n -- addr len )
    CASE
        0 OF s" Sunday"     ENDOF
        1 OF s" Monday"     ENDOF
        2 OF s" Tuesday"    ENDOF
        3 OF s" Wednesday"  ENDOF
        4 OF s" Thursday"   ENDOF
        5 OF s" Friday"     ENDOF
        6 OF s" Saturday"   ENDOF
    ENDCASE
  ;
```

ok   --

Displays the version of the FORTH language.

```
ok
\ display: uEforth
```

only   --

Reset context stack to one item, the FORTH dictionary

Non-standard, as there's no distinct ONLY vocabulary

open-blocks   addr len --

Open a block file. The default blocks file is *blocks.fb*

OPEN-FILE   addr n opt -- n

Open a file.

opt is one of the values `R/O` or `R/W` or `W/O`.

```
s" myFile" r/o open-file
```

OR   n1 n2 -- n3

Execute logic OR.

The words `AND`, `OR`, and `XOR` perform operations binary **bitwise** logic on single-precision integers at the top of the data stack.

```
0  -1    or  .  \ display 0
0  -1    or  .  \ display -1
-1   0   or  .  \ display -1
-1   -1  or  .  \ display -1
```

order   --

Print the vocabulary search order.

```
Serial
order   \ display Serial
```

over   n1 n2 -- n1 n2 n1

Place a copy of n1 on top of the stack.

```
2 5 OVER  \ duplicate 2 on top of the stack
```

page   --

Erases the screen.

PARSE   c "string" -- addr count

Parse the next word in the input stream, terminating on character c. Leave the address and character count of word. If the parse area was empty then count=0.

pause   --

Yield to other tasks.

PI   -- r

PI constant.

```
pi
F.          \ display 3.141592
\ perimeter of a circle, for r = 5.2    ---    P = 2 π R
5.2e 2e F*  pi  F*
F.          \ display 32.672560
```

pinMode   pin mode --

Set mode of GPIO.

MODE = INPUT | OUTPUT

```
04 input pinmode            \ GO4 as an input
15 input pinmode            \ G15 as an input
```

precision   -- n

Pseudo constant determining the display precision of real numbers.

Initial value 6.

If we reduce the display precision of real numbers below 6, the calculations will be when even performed with precision to 6 decimal places.

```
precision . \ display 6
pi f.       \ display 3.141592
```

```
4 set-precision
precision . \ display 4
pi f.        \ display 3.1415
```

prompt   --

Displays an interpreter availability text. Default poster:

ok

PSRAM?   -- -1|0

Stacks -1 if PSRAM memory is available.

r"   comp: -- <string> | exec: addr len

Creates a temporary counted string ended with "

R/O   -- 0

System constant. Stack 0.

R/W   -- 2

System constant. Stack 2.

r>   R: n -- S: n

Transfers n from the return stack.

This operation must always be balanced with >r

```
\ display n in binary format
: b. ( n -- )
   base @ >r
   binary .
   r> base !
  ;
```

R@   -- n

Copies the contents of the top of the return stack onto the data stack.

rdrop   S: -- R: n --

Discard top item of return stack.

READ-FILE   a n fh -- n ior

Read data from a file. The number of character actually read is returned as u2, and ior is returned 0 for a successful read.

recurse --

Append the execution semantics of the current definition to the current definition.

The usual example is the coding of the factorial function.

```
: FACTORIAL ( +n1 -- +n2)
   DUP 2 < IF DROP 1 EXIT THEN
   DUP 1- RECURSE *
;
```

remaining -- n

Indicates the remaining space for your definitions.

```
remaining .        \ display 76652
: t ;
remaining .        \ display 76632
```

remember --

Save a snapshot to the default file.

The word **REMEMBER** allows you to *freeze* the compiled code. If you compiled an application, run **REMEMBER**. Unplug the ESP32 board. Plug it back in. You should find your app.

Use **STARTUP:** to set your application's password to run on startup.

repeat --

End a indefinite loop **begin.. while.. repeat**

REPOSITION-FILE ud fileid -- ior

Set file position, and return ior=0 on success

required addr len --

Loads the contents of the file named in the character string if it has not already been loaded.

```
s" /spiffs/dumpTool.txt" required
```

reset --

Delete the default filename.

RESIZE-FILE ud fileid -- ior

Set the size of the file to ud, an unsigned double number. After using **RESIZE-FILE**, the result returned by **FILE-POSITION** may be invalid

restore   -- <:name>

Restore a snapshot from a file.

revive   --

Restore the default filename.

rm   -- "path"

Delete the file designed in file path.

rot   n1 n2 n3 -- n2 n3 n1

Rotate three values on top of stack.

rp0   -- addr

Points to the bottom of Forth's return stack.

RSHIFT   x1 u -- x2

Right shift of the value x1 by u bits.

```
64 2 rshift .  \ display 16
```

r| comp: -- <string> | exec: addr len

Creates a temporary counted string ended with |

s"   comp: -- <string> | exec: addr len

In interpretation, leaves on the data stack the string delimited by "

In compilation, compiles the string delimited by "

When executing the compiled word, returns the address and length of the string...

```
\ header for DUMP
: headDump
    s" --addr----- 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F"
  ;
headDump           \ push addr len on stack
headDump type   \ display: --addr----- 00 01 02 03 04 05 06 07 08 09 0A 0B 0C
0D 0E 0F
```

S>F   n -- r: r

Converts an integer to a real number and transfers this real to the stack of reals.

```
35 S>F
F.   \ display 35.000000
```

s>z   a n -- z

Convert a counted string string to null terminated (copies string to heap)

save   -- <:name>

Saves a snapshot of the current dictionary to a file.

save-buffers   --

Save all buffers.

SCR   -- addr

Variable pointing to the block being edited.

SDL2   --

Select SDL2 vocabulary.

see   -- name>

Decompile a FORTH definition.

```
see include
: include  bl PARSE included ;

see space
: space  bl emit ;
```

set-precision   n --

Changes the display precision of Real numbers.

The calculation precision on real numbers stops at 6 decimal places. If you request a precision greater than 6 on the decimal places of real numbers, the values displayed beyond 6 decimal places will be false.

```
pi f.    \ display 3.141592
2 set-precision
pi f.    \ display 3.14
```

SF!   r addr --

Stores a real previously deposed on the real stack at the memory address addr.

```
fvariable PRICE
3.25E PRICE SF!
```

sf,   r --

Compile a real number.

SF@   addr -- r

Get the actual number stored at address addr, usually a variable defined by `fvariable`.

```
fvariable PRICE
35.25E PRICE SF!
PRICE SF@ F.    \ display: 35.250000
```

sfloat   -- 4

Constant. value 4.

sfloat+   addr -- addr+4

Increments a memory address by the length of a real.

sfloats   n -- n*4

Calculate needed space for n reals.

sp0   -- addr

Points to the bottom of Forth's parameter stack.

SP@   -- addr

Push on stack the address of data stack.

```
\ return number cells used on stack
: stackSize ( -- n )
    SP@ SP0 - CELL/
  ;
```

space   --

Display one space.

```
\ definition of space
: space  ( -- )
    bl emit
  ;
```

spaces   n --

Displays the space character n times.

Defined since version 7.071

SPI   --

Select the `SPI` vocabulary.

List of `SPI` vocabulary words:

```
SPI.begin SPI.end SPI.setHwCs SPI.setBitOrder SPI.setDataMode
SPI.setFrequency
```

```
SPI.setClockDivider SPI.getClockDivider SPI.transfer SPI.transfer8
SPI.transfer16
```

```
SPI.transfer32 SPI.transferBytes SPI.transferBits SPI.write SPI.write16
```

```
SPI.write32 SPI.writeBytes SPI.writePixels SPI.writePattern SPI-builtins
```

startup: -- <name>

Indicates the word that should run when ESP32forth starts after initialization of the general environment.

state   -- fl

Compilation state. State can only be changed by `[` and `]`.

-1 for compiling, 0 for interpreting

str   n -- addr len

Transforms any value n into an alphanumeric string, in the current numeric base.

```
352 str type    \ display: 352
```

str=   addr1 len1 addr2 len2 -- fl

Compare two strings. Leave true if they are identical.

```
s" 123"   s" 124"
str = .       \ display 0
s" 156"   s" 156"
str= .        \ display -1
```

streams   --

Select `streams` vocabulary.

structures   --

Select the `structures` vocabulary.

```
structures
\ Information about the version of SDL in use
struct SDL_version    ( -- 3 )              \ OK 2024-11-06
    i8 field ->version-major
    i8 field ->version-minor
    i8 field ->version-patch
```

swap   n1 n2 -- n2 n1

Swaps values at the top of the stack.

```
2 5 SWAP
.     \ display 2
.     \ display 5
```

task   comp: xt dsz rsz -- <name> | exec: -- task

Create a new task with dsz size data stack and rsz size return stack running xt.

```
tasks
: hi   begin ." Time is: " ms-ticks . cr 1000 ms again ;
' hi 100 100 task my-counter
my-counter start-task hi
```

tasks   --

Select `tasks` vocabulary.

then   --

Immediate execution word used in compilation only. Mark the end a control structure of type `IF..THEN` or `IF..ELSE..THEN` .

throw   n --

Generates an error if n is not equal to zero.

If any bits of n are non-zero, pop the topmost exception frame from the exception stack, along with everything on the return stack above that frame. Then restore the input source specification in use before the corresponding CATCH and adjust the depths of all stacks defined by this standard so that they are the same as the depths saved in the exception frame (i is the same number as the i in the input arguments to the corresponding CATCH), put n on top of the data stack, and transfer control to a point just after the CATCH that pushed that exception frame.

```
: could-fail ( -- char )
   KEY DUP [CHAR] Q = IF 1 THROW THEN ;

: do-it ( a b -- c) 2DROP could-fail ;

: try-it ( --)
   1 2 ['] do-it CATCH IF
   ( x1 x2 ) 2DROP ." There was an exception" CR
   ELSE ." The character was " EMIT CR
   THEN
;

: retry-it ( -- )
   BEGIN 1 2 ['] do-it CATCH WHILE
   ( x1 x2) 2DROP ." Exception, keep trying" CR
```

```
    REPEAT ( char )
    ." The character was " EMIT CR
;
```

thru   n1 n2 --

Loads the contents of a block file, from block n1 to block n2.

tib   -- addr

returns the address of the the terminal input buffer where input text string is held.

```
tib >in @ type
\ display:
tib >in @
```

to   n --- <valname>

to assign new value to *valname*

tone   chan freq --

sets frequency freq n to channel chan.

Use ledcWriteTone

touch   -- "path"

Create "path" file if it doesn't exist.

type   addr c --

Display the string characters over c bytes.

```
: hello ( --- addr c)
s" Hello world" ;
hello type             \ display: Hello world
hello drop 5 type      \ display: Hello
```

u.   n --

Removes the value from the top of the stack and displays it as an unsigned single precision integer.

```
1 U.      \ display 1
-1 U.     \ display 18446744073709551615
```

U/MOD   u1 u2 -- rem quot

Unsigned int/int->int division.

UL@   addr -- un

Retrieve a unsigned value.

WARNING: Previous versions of ESP32forth used the word `L@`.

unloop   --

Stop a do..loop action. Using `unloop` before `exit` only in a do..loop structure.

```
: example ( -- )
   100 0 do
       cr i .
       key bl = if
           unloop exit
       then
   loop
  ;
```

until   fl --

End of `begin.. until` structure.

```
: myTestLoop ( -- )
   begin
       key dup .
       [char] A =
   until
  ;
myTestLoop \ end loop if key A pressed
```

update   --

Used for block editing. Forces the current block to the modified state.

use   -- <name>

Use "name" as the blockfile.

```
USE /spiffs/foo
```

used   -- n

Specifies the space taken up by user definitions. This includes already defined words from the FORTH dictionary.

UW@   addr -- un[2exp0..2exp16-1]

Extracts the least significant 16 bits part of a memory zone pointed to by its unsigned 32-bit address.

```
variable valX
hex 10204080 valX !
```

```
valX UW@ .      \ display 4080
valX 2 + UW@ .  \ display 1020
```

value   comp: n -- <valname> | exec: -- n

Define value.

valname leave value on stack.

A Value behaves like a Constant, but it can be changed.

```
12 value APPLES     \ Define APPLES with an initial value of 12
34 to APPLES        \ Change the value of APPLES. to is a parsing word
APPLES              \ puts 34 on the top of the stack
```

variable   comp: -- <name> | exec: -- addr

Creation word. Defines a simple precision variable.

```
variable speed
75 speed !   \ store 75 in speed
speed @ .    \ display 75
```

visual   --

Selects the `visual` vocabulary.

vlist   --

Display all words from a vocabulary.

```
Serial vlist  \ display content of Serial vocabulary
```

vocabulary   comp: -- <name> | exec: --

Definition word for a new vocabulary. In 83-STANDARD, vocabularies are no longer declared to be executed immediately.

```
\ create new vocabulary FPACK
VOCABULARY FPACK
```

W/O   -- 1

System constant. Stack 1.

web --DELETE   --

Select `web` vocabulary.

while   **fl --**

Mark the conditionnal part execution of a structure `begin..while..repeat`

```
\ logarithmus dualis of n1>0, rounded down to the next integer
: log2 ( +n1 -- n2 )
    2/ 0 begin
        over 0 >
    while
        1+ swap 2/ swap
    repeat
    nip
  ;
  7 log2 .      \ display 2
100 log2 .      \ display 6
```

windows   **--**

select `windows` vocabulary.

words   **--**

List the definition names in the first word list of the search order. The format of the display is implementation-dependent.

WRITE-FILE   **a n fh -- ior**

Write a block of memory to a file.

XOR   **n1 n2 -- n3**

Execute logic eXclusif OR.

The words `AND`, `OR`, and `XOR` perform operations binary **bitwise** logic on single-precision integers at the top of the data stack.

```
 0 -1 xor .     \ display 0
 0 -1  xor .    \ display -1
-1  0 xor .     \ display -1
-1  0  xor .    \ display 0
```

z"   **comp: -- <string> | exec: -- addr**

Compile zero terminated string into definition.

WARNING: these character strings marked with `z"` can only be used for specific functions.

z>s   **z -- a n**

Convert a null terminated string to a counted string.

[--

Enter interpretation state. **[** is an immediate word.

```
\ source for [
: [
    0 state !
    ; immediate
```

[']   comp: -- <name> | exec: -- addr

Use in compilation only. Immediate execution.

Compile the cfa of <name>

[char]   comp: -- <spaces>name | exec: -- xchar

Place xchar, the value of the first xchar of name, on the stack.

```
: GC1 [CHAR] X      ;
: GC2 [CHAR] HELLO ;
GC1 \ empile 58
GC2 \ empile 48
```

[ELSE]   --

Mark a part of conditional sequence in **[IF] ... [ELSE] ... [THEN]**.

[IF]   fl --

Begins a conditional sequence of type **[IF] ... [ELSE]** or **[IF] ... [ELSE] ... [THEN]**.

If flag is 'TRUE' do nothing (and therefore execute subsequent words as normal). If flag is 'FALSE', parse and discard words from the parse area including nested instances of **[IF]**.. **[ELSE]**.. '[THEN]' and **[IF]**.. **[THEN]** until the balancing **[ELSE]** or **[THEN]** has been parsed and discarded.

```
DEFINED? L, invert [IF]
\ compile 32 bits value in dictionnary
: L,  ( u -- )
    dup c,
    8 rshift dup c,
    8 rshift dup c,
    8 rshift dup c,
    drop
  ;
[THEN]
```

[THEN]   --

Ends a conditional sequence of type **[IF]** ... **[ELSE]** or **[IF]** ... **[ELSE]** ... **[THEN]**.

```
DEFINED? mclr [IF]
: mclr   ( mask addr -- )
    dup >r c@ swap invert and r> c!
    ;
[THEN]
```

]   --

Return to compilation. **]** is an immediate word.

With FlashForth, the words [and] allow you to use assembly code, subject to first compiling an assembler.

{   -- <names..>

Marks the start of the definition of local variables. These local variables behave like pseudo-constants.

Local variables are an interesting alternative to the manipulation of stack data. They make the code more readable.

```
: summ { n1 n2 }
    n1 n2 + . ;
3 5 summ    \ display 8
```

graphics

color   -- n

Definie color. Default value: 0

```
\ Pen in red color:
$ff0000 to color    \ $rrggbb
```

event   -- 0

Constant. Default Value 0

EXPOSED   -- 2

Constant. Value 2

FINISHED   -- 7

Constant. Value 7

height   -- 0

Value. Default Value 0

IDLE   -- 0

Constant. Value 0

last-char   -- 0

Constant. Default Value 0

last-key   -- 0

Constant. Default Value 0

LEFT-BUTTON   -- 255

Constant. Value 255

MIDDLE-BUTTON   -- 254

Constant. Value 254

MOTION   -- 3

Constant. Value 3

mouse-x   -- **0**

Constant. Default Value 0

mouse-y   -- **0**

Constant. Default Value 0

pixel   **w h --**

Draws a pixel in position w h

PRESSED   -- **4**

Constant. Value 4

RELEASED   -- **5**

Constant. Value 5

RESIZED   -- **1**

Constant. Value 1

RIGHT-BUTTON   -- **253**

Constant. Value 253

TYPED   -- **6**

Constant. Value 6

width   -- **0**

Value. Default Value 0

streams

>stream　addr len stream --

Store a string characters in a stream.

```
streams
1000 stream myStream
s" this is " myStream >stream
s" a test."  myStream >stream
\ now, myStream content is: "this is a test."
```

ch>stream　c stream --

add character c to a stream.

```
streams
1000 stream myStream
s" this is" myStream >stream
$0d myStream ch>stream
$0a myStream ch>stream
s" a test" myStream >stream

myStream dup
    0 swap >offset
    swap cell + @
    type
\ display:
\ this is
\ a test.
```

empty?　-- fl

Push -1 if stream is empty, otherwise push 0.

full?　-- fl

Push -1 if stream is full, otherwise push 0.

stream　comp: n -- <name> | exec: -- addr

Create a memory space of n characters.

```
200 stream input-stream
```

stream#　sz -- n

Used bye **full?** and **empty?**.

stream>ch　addr -- c

Fetch a character from stream.

structures

field   comp: n -- <:name>

Definition word for a new field in a structure.

```
also structures
struct esp_partition_t
  ( Work around changing struct layout )
  esp_partition_t_size 40 >= [IF]
    ptr field p>gap
  [THEN]
  ptr field p>type
  ptr field p>subtype
  ptr field p>address
  ptr field p>size
  ptr field p>label
```

i16   -- 2

Pseudo constant defined by `typer`. At runtime, drops the size of the datatype and puts a copy of that size in the `last-align` variable

i32   -- 4

Pseudo constant defined by `typer`. At runtime, drops the size of the datatype and puts a copy of that size in the `last-align` variable

i64   -- 8

Pseudo constant defined by `typer`. At runtime, drops the size of the datatype and puts a copy of that size in the `last-align` variable

i8   -- 1

Pseudo constant defined by `typer`. At runtime, drops the size of the datatype and puts a copy of that size in the `last-align` variable

last-struct   -- addr

Variable pointing to the last defined structure.

long   -- 4

Pseudo constant defined by `typer`. At runtime, drops the size of the datatype and puts a copy of that size in the `last-align` variable

ptr   -- 4

Pseudo constant defined by `typer`. At runtime, drops the size of the datatype and puts a copy of that size in the `last-align` variable

struct   comp: -- <:name>

Definition word for structures.

```
also structures
struct esp_partition_t
```

typer   comp: n1 n2 -- <name> | exec: -- n

Definition word for `i8 i16 i32 i64 ptr long`

tasks

.tasks   --

Display list active tasks.

```
.tasks  \ display: main-task
```

main-task   -- task

Main task. Leave pointer task on stack

task-list   -- addr

Variable. Point to tasks list.

windows

ANSI_FIXED_FONT   -- n

Constant, value: $8000000b

ANSI_VAR_FONT   -- n

Constant, value: $8000000c

BI_RGB   -- n

Constant, value: 0

BLACK_BRUSH   -- n

Constant, value: $80000004

BLACK_PEN   -- n

Constant, value: $80000007

BM_CLICK   -- 245

Constant. Value 245

Used by **WM_>name**

BM_GETCHECK   -- 240

Constant. Value 240

Used by **WM_>name**

Gets the check state of a radio button or check box.

BM_GETIMAGE   -- 246

Constant. Value 246

Used by **WM_>name**

BM_GETSTATE   -- 242

Constant. Value 242

Used by **WM_>name**

Retrieves the state of a button or check box.

BM_SETCHECK   -- 241

Constant. Value 241

Used by **WM_>name**

Sets the check state of a radio button or check box.

BM_SETDONTCLICK   -- 248

Constant. Value 248

Used by **WM_>name**

BM_SETIMAGE   -- 247

Constant. Value 247

Used by **WM_>name**

BM_SETSTYLE   -- 244

Constant. Value 244

Used by **WM_>name**

CB_ADDSTRING   -- 323

Constant. Value 323

Used by **WM_>name**

Adds a string to the list box of a combo box. If the combo box does not have the CBS_SORT style, the string is added to the end of the list. Otherwise, the string is inserted into the list, and the list is sorted.

CB_FINDSTRING   -- 332

Constant. Value 332

Used by **WM_>name**

Searches the list box of a combo box for an item beginning with the characters in a specified string.

CB_FINDSTRINGEXACT   -- 344

Constant. Value 344

Used by **WM_>name**

CB_GETCOMBOBOXINFO   -- 356

Constant. Value 356

Used by **WM_>name**

CB_GETCOUNT   -- 326

Constant. Value 326

Used by **WM_>name**

Gets the number of items in the list box of a combo box.

CB_GETCURSEL   -- 327

Constant. Value 327

Used by **WM_>name**

An application sends a CB_GETCURSEL message to retrieve the index of the currently selected item, if any, in the list box of a combo box.

CB_GETDROPPEDCONTROLRECT   -- 338

Constant. Value 338

Used by **WM_>name**

CB_GETDROPPEDSTATE   -- 343

Constant. Value 343

Used by **WM_>name**

CB_GETDROPPEDWIDTH   -- 351

Constant. Value 351

Used by **WM_>name**

CB_GETEDITSEL   -- 320

Constant. Value 320

Used by **WM_>name**

CB_GETEXTENDEDUI   -- 342

Constant. Value 342

Used by **WM_>name**

CB_GETHORIZONTALEXTENT   -- 349

Constant. Value 349

Used by **WM_>name**

CB_GETITEMDATA -- 336

Constant. Value 336

Used by WM_>name

CB_GETITEMHEIGHT -- 340

Constant. Value 340

Used by WM_>name

CB_GETLBTEXT -- 328

Constant. Value 328

Used by WM_>name

CB_GETLBTEXTLEN -- 329

Constant. Value 329

Used by WM_>name

CB_GETLOCALE -- 346

Constant. Value 346

Used by WM_>name

CB_GETTOPINDEX -- 347

Constant. Value 347

Used by WM_>name

CB_INITSTORAGE -- 353

Constant. Value 353

Used by WM_>name

CB_INSERTSTRING -- 330

Constant. Value 330

Used by WM_>name

CB_LIMITTEXT -- 321

Constant. Value 321

Used by WM_>name

CB_MSGMAX   -- 357

Constant. Value 357

Used by **WM_>name**

CB_MULTIPLEADDSTRING   -- 355

Constant. Value 355

Used by **WM_>name**

CB_RESETCONTENT   -- 331

Constant. Value 331

Used by **WM_>name**

CB_SELECTSTRING   -- 333

Constant. Value 333

Used by **WM_>name**

CB_SETCURSEL   -- 334

Constant. Value 334

Used by **WM_>name**

CB_SETDROPPEDWIDTH   -- 352

Constant. Value 352

Used by **WM_>name**

CB_SETEDITSEL   -- 322

Constant. Value 322

Used by **WM_>name**

CB_SETEXTENDEDUI   -- 341

Constant. Value 341

Used by **WM_>name**

CB_SETHORIZONTALEXTENT   -- 350

Constant. Value 350

Used by **WM_>name**

CB_SETITEMDATA -- 337

Constant. Value 337

Used by `WM_>name`

CB_SETITEMHEIGHT -- 339

Constant. Value 339

Used by `WM_>name`

CB_SETLOCALE -- 345

Constant. Value 345

Used by `WM_>name`

CB_SETTOPINDEX -- 348

Constant. Value 348

Used by `WM_>name`

CB_SHOWDROPDOWN -- 335

Constant. Value 335

Used by `WM_>name`

console-started -- 0

Value initialized to zero.

Used by `init-console`

DC_BRUSH -- n

Constant, value: $80000012

DC_PEN -- n

Constant, value: $80000013

DEFAULT_GUI_FONT -- n

Constant, value: $80000011

DEFAULT_PALETTE -- n

Constant, value: $8000000f

DEVICE_DEFAULT_PALETTE   -- n

Constant, value: $8000000e

DISABLE_NEWLINE_AUTO_RETURN   -- n

Constant. Value $0008

DKGRAY_BRUSH   -- n

Constant, value: $80000003

dll   comp: zStr -- <:name>

Creates an access ticket to a Windows library.

```
z" Kernel32.dll" dll Kernel32
```

EM_CHARFROMPOS   -- 215

Constant. Value 215

Used by WM_>name

EM_EMPTYUNDOBUFFER   -- 205

Constant. Value 205

Used by WM_>name

EM_FMTLINES   -- 200

Constant. Value 200

Used by WM_>name

EM_GETFIRSTVISIBLELINE   -- 206

Constant. Value 206

Used by WM_>name

EM_GETIMESTATUS   -- 217

Constant. Value 217

Used by WM_>name

EM_GETLIMITTEXT   -- 213

Constant. Value 213

Used by WM_>name

EM_GETMARGINS   -- 212

Constant. Value 212

Used by **WM_>name**

EM_GETPASSWORDCHAR   -- 210

Constant. Value 210

Used by **WM_>name**

EM_GETWORDBREAKPROC   -- 209

Constant. Value 209

Used by **WM_>name**

EM_LINEFROMCHAR   -- 201

Constant. Value 201

Used by **WM_>name**

EM_POSFROMCHAR   -- 214

Constant. Value 214

Used by **WM_>name**

EM_SETIMESTATUS   -- 216

Constant. Value 216

Used by **WM_>name**

EM_SETMARGINS   -- 211

Constant. Value 211

Used by **WM_>name**

EM_SETPASSWORDCHAR   -- 204

Constant. Value 204

Used by **WM_>name**

EM_SETREADONLY   -- 207

Constant. Value 207

Used by **WM_>name**

EM_SETTABSTOPS -- 203

Constant. Value 203

Used by `WM_>name`

EM_SETWORDBREAK -- 202

Constant. Value 202

Used by `WM_>name`

EM_SETWORDBREAKPROC -- 209

Constant. Value 209

Used by `WM_>name`

EM_UNDO -- 199

Constant. Value 199

Used by `WM_>name`

ENABLE_INSERT_MODE -- n

Constant, value: $0020

ENABLE_PROCESSED_INPUT -- n

Constant, value: $0001

init-console --

Initializes the Windows console.

Kernel32 --

Word defined by `dll`.

Then allows access to the functions of **Kernel32.dll**

MALLOC_CAP_32BIT -- 2

Constant. Value 2

MALLOC_CAP_8BIT -- 4

Constant. Value 4

MALLOC_CAP_DMA -- 8

Constant. Value 8

MALLOC_CAP_EXEC   -- 1

Constant. Value 1

NULL_BRUSH   -- n

Constant. Value $80000005

SBM_ENABLE_ARROWS   -- 228

Constant. Value 228

Used by WM_>name

SBM_GETPOS   -- 225

Constant. Value 225

Used by WM_>name

SBM_GETRANGE   -- 227

Constant. Value 227

Used by WM_>name

SBM_GETSCROLLBARINFO   -- 235

Constant. Value 235

Used by WM_>name

SBM_GETSCROLLINFO   -- 234

Constant. Value 234

Used by WM_>name

SBM_SETPOS   -- 224

Constant. Value 224

Used by WM_>name

SBM_SETRANGE   -- 226

Constant. Value 226

Used by WM_>name

SBM_SETRANGEREDRAW   -- 230

Constant. Value 230

Used by `WM_>name`

SBM_SETSCROLLINFO   -- 233

Constant. Value 233

Used by `WM_>name`

stdin   -- 0

Value initialized to zero.

Used by `init-console`

stdout   -- 0

Value initialized to zero.

Used by `init-console`

win-type   addr len --

Dispaly string on windows console

windows-builtins   -- n

Vocabulary entry point `windows`

WM_>name   msg -- a n

Extracts the address in length from the header corresponding to the Windows message between `WM_PENWINLAST` and `WM_NULL`

WM_ACTIVATE   -- 6

Constant. Value 6

Used by `WM_>name`

WM_AFXFIRST   -- 864

Constante. value 864.

Use by `WM_>name`

WM_AFXLAST   -- 896

Constante. value 895.

Use by `WM_>name`

WM_APPCOMMAND -- 793

Constante. value 793.

Use by `WM_>name`

WM_CHANGECBCHAIN -- 781

Constant. Value 781

WM_CHAR -- 258

stack 258.

Used by `WM_>name`

WM_CLEAR -- 771

Constant. Value 771

Used by `WM_>name`

WM_COPY -- 769

Constant. Value 769

Used by `WM_>name`

WM_CREATE -- 1

stack 1.

WM_CUT -- 768

Constant. Value 768

Used by `WM_>name`

WM_DEADCHAR -- 259

stack 259.

WM_DESTROY -- 2

Constant. Value 2

Used by `WM_>name`

WM_DESTROYCLIPBOARD -- 775

Constant. Value 775

Used by `WM_>name`

WM_DRAWCLIPBOARD   -- 776

Constant. Value 776

Used by `WM_>name`

WM_ENABLE   -- 10

Constant. Value 10

Used by `WM_>name`

WM_ENTERIDLE   -- 289

Constant. Value 289

Used by `WM_>name`

WM_GETTEXT   -- 13

Constant. Value 13

Used by `WM_>name`

WM_GLOBALRCCHANGE   -- 899

Constante. value 899.

Use by `WM_>name`

WM_HANDHELDFIRST   -- 856

Constante. value 856.

Use by `WM_>name`

WM_HANDHELDLAST   -- 863

Constante. value 863.

Use by `WM_>name`

WM_HEDITCTL   -- 901

Constant. Value 901

Used by `WM_>name`

WM_HOOKRCRESULT   -- 898

Constante. value 898.

Use by `WM_>name`

WM_HOTKEY   -- 786

Constant. Value 786

WM_HSCROLL   -- 276

Constant. Value 276

Used by **WM_>name**

WM_HSCROLLCLIPBOARD   -- 782

Constant. Value 782

WM_IMEKEYDOWN   -- 656

Constant. Value 656

Used by **WM_>name**

WM_IMEKEYUP   -- 657

Constant. Value 657

Used by **WM_>name**

WM_IME_CHAR   -- 646

Constant. Value 646

Used by **WM_>name**

WM_IME_COMPOSITIONFULL   -- 644

Constant. Value 644

Used by **WM_>name**

WM_IME_CONTROL   -- 643

Constant. Value 643

Used by **WM_>name**

WM_IME_KEYDOWN   -- 656

Constant. Value 656

Used by **WM_>name**

WM_IME_KEYUP   -- 657

Constant. Value 657

Used by `WM_>name`

WM_IME_NOTIFY   -- 642

Constant. Value 642

Used by `WM_>name`

WM_IME_REPORT   -- 640

Constant. Value 640

Used by `WM_>name`

WM_IME_REQUEST   -- 648

Constant. Value 648

Used by `WM_>name`

WM_IME_SELECT   -- 645

Constant. Value 645

Used by `WM_>name`

WM_IME_SETCONTEXT   -- 641

Constant. Value 641

Used by `WM_>name`

WM_INITMENU   -- 278

Constant. Value 278

Used by `WM_>name`

WM_INITMENUPOPUP   -- 279

Constant. Value 279

Used by `WM_>name`

WM_INPUT   -- 255

Constant. Value 255

Used by `WM_>name`

WM_KEYDOWN   -- 256

Constant. Value 256

Used by `WM_>name`

WM_KEYUP   -- 257

Constant. Value 257

Used by `WM_>name`

WM_KILLFOCUS   -- 0

Constant. Value 0

WM_LBUTTONDBLCLK   -- 515

Constant. Value 515

Used by `WM_>name`

WM_LBUTTONDOWN   -- 513

Constant. Value 513

Used by `WM_>name`

WM_LBUTTONUP   -- 514

Constant. Value 514

Used by `WM_>name`

WM_MBUTTONDBLCLK   -- 521

Constant. Value 521

Used by `WM_>name`

WM_MBUTTONDOWN   -- 519

Constant. Value 519

Used by `WM_>name`

WM_MENUCHAR   -- 288

Constant. Value 288

Used by `WM_>name`

WM_MENUSELECT   -- 287

Constant. Value 287

Used by `WM_>name`

WM_MOUSEFIRST   -- 512

Constant. Value 512

Used by `WM_>name`

WM_MOUSEHOVER   -- 673

Constant. Value 673

Used by `WM_>name`

WM_MOUSELAST   -- 521

Constant. Value 521

Used by `WM_>name`

WM_MOUSELEAVE   -- 675

Constant. Value 675

Used by `WM_>name`

WM_MOUSEMOVE   -- 512

Constant. Value 512

Used by `WM_>name`

WM_MOVE   -- 3

Constant. Value 3

Used by `WM_>name`

WM_NCMOUSEHOVER   -- 672

Constant. Value 672

Used by `WM_>name`

WM_NCMOUSELEAVE   -- 674

Constant. Value 674

Used by `WM_>name`

WM_NULL   -- 0

Constant. Value 0

WM_PAINTCLIPBOARD   -- 777

Constant. Value 777

Used by **WM_>name**

WM_PALETTECHANGED   -- 785

Constant. Value 785

WM_PALETTEISCHANGING   -- 784

Constant. Value 784

WM_PASTE   -- 770

Constant. Value 770

Used by **WM_>name**

WM_PENCTL   -- 901

Constant. Value 901

Used by **WM_>name**

WM_PENEVENT   -- 904

Constant. Value 904

Used by **WM_>name**

WM_PENMISC   -- 902

Constant. Value 902

Used by **WM_>name**

WM_PENMISCINFO   -- 899

Constante. value 899.

Use by **WM_>name**

WM_PENWINFIRST   -- 896

Constante. value 896.

Use by **WM_>name**

WM_PENWINLAST   -- 911

Constant. Value 911

Used by `WM_>name`

Value used to set an upper limit for Pen Windows (PenWin) style messages.

WM_PRINTCLIENT   -- 792

Constant. Value 792

WM_QUERYNEWPALETTE   -- 783

Constant. Value 783

WM_RBUTTONDBLCLK   -- 518

Constant. Value 518

Used by `WM_>name`

WM_RBUTTONDOWN   -- 516

Constant. Value 516

Used by `WM_>name`

WM_RBUTTONUP   -- 517

Constant. Value 517

Used by `WM_>name`

WM_RCRESULT   -- 898

Constante. value 897.

Use by `WM_>name`

WM_RENDERALLFORMATS   -- 774

Constant. Value 774

Used by `WM_>name`

WM_RENDERFORMAT   -- 774

Constant. Value 773

Used by `WM_>name`

WM_SETFOCUS   -- 7

Constant. Value 7

WM_SETREDRAW -- 11

Constant. Value 11

WM_SETTEXT -- 12

Constant. Value 12

Used by **WM_>name**

WM_SIZE -- 5

Constant. Value 5

Used by **WM_>name**

WM_SKB -- 900

Constant. Value 900

Used by **WM_>name**

WM_SYSDEADCHAR -- 258

stack 263.

WM_SYSTIMER -- 280

Constant. Value 280

Used by **WM_>name**

WM_UNDO -- 772

Constant. Value 772

Used by **WM_>name**

WM_VSCROLL -- 277

Constant. Value 277

Used by **WM_>name**

WM_VSCROLLCLIPBOARD -- 778

Constant. Value 778

Mots FORTH par utilisation

arithmetic integer

* (n1 n2 -- n3)
*/ (n1 n2 n3 -- n4)
*/MOD (n1 n2 n3 -- n4 n5)
\+ (n1 n2 -- n3)
\- (n1 n2 -- n1-n2)
/mod (n1 n2 -- n3 n4)
1+ (n -- n+1)
1- (n -- n-1)
2* (n -- n*2)
2/ (n -- n/2)
4* (n -- n*4)
4/ (n -- n/4)
ARSHIFT (x1 u -- x2)
mod (n1 n2 -- n3)
negate (n -- -n')

arithmetic real

#f+s (r:r)
1/F (r -- r')
F* (r1 r2 -- r3)
F** (r_val r_exp -- r)
F+ (r1 r2 -- r3)
F- (r1 r2 -- r3)
F/ (r1 r2 -- r3)
F0< (r -- fl)
F0= (r -- fl)
F>S (r -- n)
FABS (r1 -- r1')
FATAN2 (r-tan -- r-rad)
fconstant (comp: r -- <name> | exec: --
r)
FCOS (r1 -- r2)
FEXP (ln-r -- r)
FLN (r -- ln-r)
FLOOR (r1 -- r2)
FMAX (r1 r2 -- r1|r2)
FMIN (r1 r2 -- r1|r2)

FNEGATE (r1 -- r1')
FSIN (r1 -- r2)
FSINCOS (r1 -- rcos rsin)
fsqrt (r1 -- r2)
pi (-- r)
S>F (n -- r: r)

block edit list

a (n --)
copy (from to --)
d (n --)
e (n --)
editor (--)
flush (--)
list (n --)
load (n --)
n (--)
open-blocks (addr len --)
p (--)
r (n --)
thru (n1 n2 --)
update (--)
use (-- <name>)
wipe (--)

chars strings

(n1 -- n2)
#FS (r:r --)
#s (n1 -- n=0)
<# (n --)
extract (n base -- n c)
F>NUMBER? (addr len -- real:r fl)
hold (c --)
r| (comp: -- <string> | exec: addr len)
s" (comp: -- <string> | exec: addr len)
s>z (a n -- z)
str (n -- addr len)
str= (addr1 len1 addr2 len2 -- fl)
z" (comp: -- <string> | exec: -- addr)
z>s (z -- a n)
[char] (comp: -- name | exec: -- xchar)

comparaison logical

0< (x1 --- fl)
0<> (n -- fl)
0= (x -- fl)
< (n1 n2 -- fl)
<= (n1 n2 -- fl)
<> (x1 x2 -- fl)
= (n1 n2 -- fl)
> (x1 x2 -- fl)
>= (x1 x2 -- fl)
f< (r1 r2 -- fl)
f<= (r1 r2 -- fl)
f<> (r1 r2 -- fl)
f= (r1 r2 -- fl)
f> (r1 r2 -- fl)
f>= (r1 r2 -- fl)
invert (x1 -- x2)
max (n1 n2 -- n1|n2)
min (n1 n2 -- n1|n2)
OR (n1 n2 -- n3)
XOR (n1 n2 -- n3)

definition words

: (comp: -- <word> | exec: --)
:noname (-- cfa-addr)
; (--)
constant (comp: n -- <name> | exec: -- n)
CREATE (comp: -- <name> | exec: -- addr)
defer (-- <vec-name>)
DOES> (comp: -- | exec: -- addr)
fvariable (comp: -- <name> | exec: -- addr)
value (comp: n -- <valname> | exec: -- n)
variable (comp: -- <name> | exec: -- addr)
vocabulary (comp: -- <name> | exec: --)

display

. (n --)
." (-- <string>)
.s (--)
? (addr -- c)
cr (--)
emit (x --)
esc (--)
f. (r --)
f.s (--)
ip. (--)
n. (n --)
normal (--)
ok (--)
prompt (--)
see (-- name>)
space (--)
spaces (n --)
type (addr c --)
u. (n --)
vlist (--)
words (--)

files words

BIN (mode -- mode')
block (n -- addr)
block-fid (-- n)
block-id (-- n)
cat (-- <path>)
CLOSE-FILE (fileid -- ior)
common-default-use (--)
cp (-- "src" "dst")
CREATE-FILE (a n mode -- fh ior)
DELETE-FILE (a n -- ior)
dump-file (addr len addr2 len2 --)
edit (-- <filename>)
file-exists? (addr len --)
FILE-POSITION (fileid -- ud ior)
FILE-SIZE (fileid -- ud ior)
FLUSH-FILE (fileid –- ior)
include (-- <:name>)
included? (addr len -- f)
ls (-- "path")
mv (-- "src" "dest")
OPEN-FILE (addr n opt -- n)
R/O (-- 0)
R/W (-- 2)
READ-FILE (a n fh -- n ior)
REPOSITION-FILE (ud fileid -- ior)
required (addr len --)
RESIZE-FILE (ud fileid -- ior)
rm (-- "path")
save-buffers (--)
touch (-- "path")
W/O (-- 1)
WRITE-FILE (a n fh -- ior)

loop and branch

+loop (n --)
?do (n1 n2 --)
aft (--)
begin (--)
CASE (--)
else (--)
ENDCASE (--)
ENDOF (--)
for (n --)
if (fl --)
loop (--)
next (--)
OF (n --)
repeat (--)
then (--)
unloop (--)
until (fl --)
while (fl --)
[ELSE] (--)
[IF] (fl --)
[THEN] (--)

memory access

! (n addr --)
2! (n1 n2 addr --)
2@ (addr -- d)
@ (addr -- n)
c! (c addr --)
c@ (addr -- c)
FP@ (-- addr)
m! (val shift mask addr --)
m@ (shift mask addr -- val)
UL@ (addr -- un)
UW@ (addr -- un[2exp0..2exp16-1])

stack manipulation

-rot (n1 n2 n3 -- n3 n1 n2)

2drop (n1 n2 n3 n4 -- n1 n2)

2dup (n1 n2 -- n1 n2 n1 n2)

>r (S: n -- R: n)

?dup (n -- n | n n)

drop (n --)

dup (n -- n n)

FDROP (r1 --)

FDUP (r1 -- r1 r1)

FNIP (r1 r2 -- r2)

FOVER (r1 r2 -- r1 r2 r1)

FSWAP (r1 r2 -- r1 r2)

nip (n1 n2 -- n2)

over (n1 n2 -- n1 n2 n1)

r> (R: n -- S: n)

R@ (-- n)

rdrop (S: -- R: n --)

swap (n1 n2 -- n2 n1)