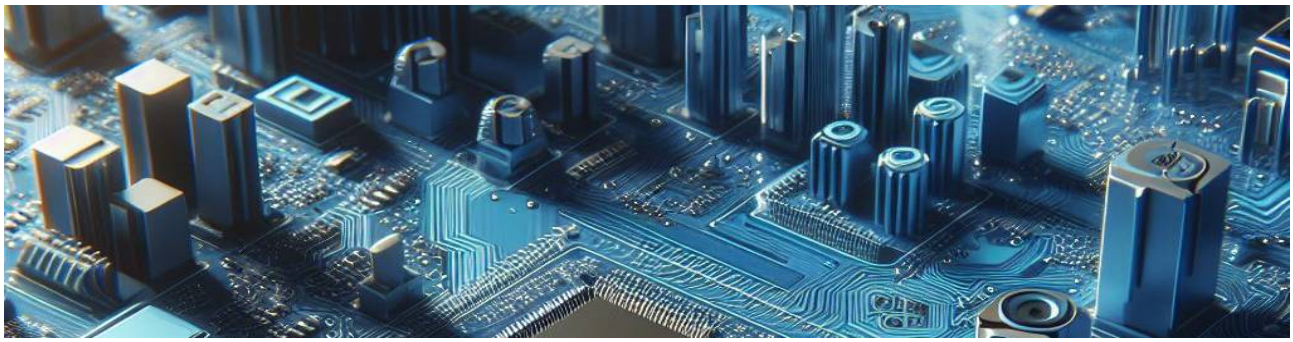
eForth Windows

Reference manual

version 1.0 - 30 novembre 2023



Author

- Marc PETREMANN petremann@arduino-forth.com

Content

FORTH words by usage

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)

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 d1 -- d=0

Converts the rest of d1 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`.

```
defer xEmit
: vxEmit ( c ---)
    1+ emit ;
' vxEmit is xEmit
```

'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" Saterday" ;
: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 followed 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.

ARSHIFT x1 u -- x2

Arithmetic right shift of u

asm --

Select the `asm` vocabulary.

assembler --

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`.

Execute by default `esp32-bye` (in voc. Internals).

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 /spiffs/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 -- 4

Return number of bytes in a 32-bit integer.

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.

```
binary  \ masks are 32 bit length
00000000000000000000000000000100 constant ledGREEN      \  green LED on G2
```

```
00000000001000000000000000000000 constant ledYELLOW     \ yellow LED on G21
00000000000000100000000000000000 constant ledRED        \    red LED on G17
decimal
```

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 `serial-key`.

default-key? -- fl

Execute `serial-key?`.

default-type addr len --

Execute `serial-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.

```
defer xEmit
: vxEmit ( c ---)
    1+ emit ;
' vxEmit is xEmit
```

DEFINED? -- <word>

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

```
DEFINED FORGET      \ push non null value on stack
DEFINED LotusBlue   \ push 0 value on stack if LotusBlue don't defined

\ 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
```

DELETE-FILE a n -- ior

Delete a named file from disk, and return ior=0 on success.

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

This version is not very interesting. Prefer this version:

DUMP tool for ESP32Forth

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.

```
: serial2-type ( a n -- )
    Serial2.write drop ;

: typeToLoRa ( -- )
    0 echo !    \ disable display echo from terminal
    ['] serial2-type is type
  ;

: typeToTerm ( -- )
```

```
    ['] default-type is type
    -1 echo !    \ enable display echo from terminal
  ;
```

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.

ESP32-C3? -- -1|0

Stacks -1 if the card is ESP32-C3.

ESP32-S2? -- -1|0

Stacks -1 if the card is ESP32-S2.

ESP32-S3? -- -1|0

Stacks -1 if the card is ESP32-S3.

ESP32? -- -1|0

Stacks -1 if the card is ESP32.

evaluate addr len --

Evaluate the content of a string.

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

EXECUTE addr --

Execute word at addr.

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 ESP32Forth's reals stack (data 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.

INPUT -- 1

Constant. Value 1. Defines the direction of use of a GPIO register as an input.

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.

```
defer xEmit
: vxEmit ( c ---)
    1+ emit ;
' vxEmit is xEmit
```

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.

```
hex
3ff44004 constant GPIO_OUT_REG

: led-off  ( -- )
    0 GPIO_OUT_REG 1!
    ;

: led-on  ( -- )
    4 GPIO_OUT_REG 1!
    ;
```

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.

```
256 string LoRaRX
s" +RCV=55,27,this is a transmission test,-36,40" LoRaRX $!

: scan$ ( char addr len -- )
    0 do
        2dup i + c@ = if
            i cr .
            leave
        then
    loop
    2drop
  ;


char , LoRaRX scan$
```

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.

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 \ detay 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`

nip n1 n2 -- n2

Remove n1 from the stack.

nl -- 10

Value 10 on stack.

normal --

Disables selected colors for display.

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 ESP32forth language.

```
ok
\ display: ESP32forth v7.0.6.10 - rev 17c8b34289028a5c731d
```

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 (./myforth or /spiffs/myforth on ESP32).

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 (data 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.

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.

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

SF! r addr --

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

sf, r --

Compile a real number.

SF@ addr -- r

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

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 (data 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.

Here we have defined the word `myBoot` which displays a text on startup.

To test the correct execution, you can type `bye` , which restart ESP32forth.

You can also unplug the ESP32 board and plug it back in. This is this test that was carried out. Here is the result in the terminal.

```
: myBoot ( -- )
    ." This is a text displayed from boot" ;
startup: myBoot
```

```
\ on restart:
--> This is a text displayed from bootESP32forth v7.0.5 - rev
33cf8aaa6fe3e0bc4a
bf3e4cd5c496a3071b9171
 ok
 ok
```

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.

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.

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
```

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 65535
```

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 --

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>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>

```
serial \ Select Serial vocabulary

: serial2-type ( a n -- )
    Serial2.write drop ;

: typeToLoRa ( -- )
    0 echo !    \ disable display echo from terminal
    ['] serial2-type is type
  ;

: typeToTerm ( -- )
    ['] default-type is type
    -1 echo !   \ enable display echo from terminal
  ;
```

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

```
\ Load constant $1234 to top of stack
: a-number ( -- 1234 )
    dup                  \ Make space for new TOS value
    [ R24 $34 ldi, ]
    [ R25 $12 ldi, ]
```

```
;
```

{ -- <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

Words defined in `graphics` vocabulary.

```
flip poll wait window heart vertical-flip viewport scale translate }g g{
screen>g box color pressed? pixel height width event last-char last-key
mouse-y mouse-x RIGHT-BUTTON MIDDLE-BUTTON LEFT-BUTTON FINISHED TYPED RELEASED
PRESSED MOTION EXPOSED RESIZED IDLE internals
```

graphics → internals

Words defined in `graphics → internals` vocabulary.

```
GrfWindowProc msg>pressed msg>button rescale binfo msgbuf ps hdc hwnd GrfClass
hinstance GrfWindowTitle GrfClassName raw-heart heart-ratio heart-initialize
cmax! cmin! heart-end heart-start heart-size heart-steps heart-f raw-box
g> >g gp gstack hline ty tx sy sx key-state! key-state key-count backbuffer
```

windows

Words defined in `windows` vocabulary.

```
WM_>name WM_PENWINLAST WM_PENEVENT WM_CTLINIT WM_PENMISC WM_PENCTL WM_HEDITCTL
WM_SKB WM_PENMISCINFO WM_GLOBALRCCHANGE WM_HOOKRCRESULT WM_RCRESULT WM_PENWINFIRST
WM_AFXLAST WM_AFXFIRST WM_HANDHELDLAST WM_HANDHELDFIRST WM_APPCOMMAND WM_PRINTCLIENT
WM_PRINT WM_HOTKEY WM_PALETTECHANGED WM_PALETTEISCHANGING WM_QUERYNEWPALETTE
WM_HSCROLLCLIPBOARD WM_CHANGECBCHAIN WM_ASKCBFORMATNAME WM_SIZECLIPBOARD
WM_VSCROLLCLIPBOARD WM_PAINTCLIPBOARD WM_DRAWCLIPBOARD WM_DESTROYCLIPBOARD
WM_RENDERALLFORMATS WM_RENDERFORMAT WM_UNDO WM_CLEAR WM_PASTE WM_COPY WM_CUT
WM_MOUSELEAVE WM_NCMOUSELEAVE WM_MOUSEHOVER WM_NCMOUSEHOVER WM_IME_KEYUP
WM_IMEKEYUP WM_IME_KEYDOWN WM_IMEKEYDOWN WM_IME_REQUEST WM_IME_CHAR WM_IME_SELECT
WM_IME_COMPOSITIONFULL WM_IME_CONTROL WM_IME_NOTIFY WM_IME_SETCONTEXT WM_IME_REPORT
WM_MDIREFRESHMENU WM_DROPFILES WM_EXITSIZEMOVE WM_ENTERSIZEMOVE WM_MDISETMENU
WM_MDIGETACTIVE WM_MDIICONARRANGE WM_MDICASCADE WM_MDITILE WM_MDIMAXIMIZE
WM_MDINEXT WM_MDIRESTORE WM_MDIACTIVATE WM_MDIDESTROY WM_MDICREATE WM_DEVICECHANGE
WM_POWERBROADCAST WM_MOVING WM_CAPTURECHANGED WM_SIZING WM_NEXTMENU WM_EXITMENULOOP
WM_ENTERMENULOOP WM_PARENTNOTIFY WM_MOUSEHWHEEL WM_XBUTTONDBLCLK WM_XBUTTONUP
WM_XBUTTONDOWN WM_MOUSEWHEEL WM_MOUSELAST WM_MBUTTONDBLCLK WM_MBUTTONUP
WM_MBUTTONDOWN WM_RBUTTONDBLCLK WM_RBUTTONUP WM_RBUTTONDOWN WM_LBUTTONDBLCLK
WM_LBUTTONUP WM_LBUTTONDOWN WM_MOUSEMOVE WM_MOUSEFIRST CB_MSGMAX CB_GETCOMBOBOXINFO
CB_MULTIPLEADDSTRING CB_INITSTORAGE CB_SETDROPPEDWIDTH CB_GETDROPPEDWIDTH
CB_SETHORIZONTALEXTENT CB_GETHORIZONTALEXTENT CB_SETTOPINDEX CB_GETTOPINDEX
CB_GETLOCALE CB_SETLOCALE CB_FINDSTRINGEXACT CB_GETDROPPEDSTATE CB_GETEXTENDEDUI
CB_SETEXTENDEDUI CB_GETITEMHEIGHT CB_SETITEMHEIGHT CB_GETDROPPEDCONTROLRECT
CB_SETITEMDATA CB_GETITEMDATA CB_SHOWDROPDOWN CB_SETCURSEL CB_SELECTSTRING
CB_FINDSTRING CB_RESETCONTENT CB_INSERTSTRING CB_GETLBTEXTLEN CB_GETLBTEXT
CB_GETCURSEL CB_GETCOUNT CB_DIR CB_DELETESTRING CB_ADDSTRING CB_SETEDITSEL
CB_LIMITTEXT CB_GETEDITSEL WM_CTLCOLORSTATIC WM_CTLCOLORSCROLLBAR WM_CTLCOLORDLG
WM_CTLCOLORBTN WM_CTLCOLORLISTBOX WM_CTLCOLOREDIT WM_CTLCOLORMSGBOX WM_LBTRACKPOINT
WM_QUERYUISTATE WM_UPDATEUISTATE WM_CHANGEUISTATE WM_MENUCOMMAND WM_UNINITMENUPOPUP
WM_MENUGETOBJECT WM_MENUDRAG WM_MENURBUTTONUP WM_ENTERIDLE WM_MENUCHAR
WM_MENUSELECT WM_SYSTIMER WM_INITMENUPOPUP WM_INITMENU WM_VSCROLL WM_HSCROLL
WM_TIMER WM_SYSCOMMAND WM_COMMAND WM_INITDIALOG WM_IME_KEYLAST WM_IME_COMPOSITION
WM_IME_ENDCOMPOSITION WM_IME_STARTCOMPOSITION WM_INTERIM WM_CONVERTRESULT
WM_CONVERTREQUEST WM_KEYLAST WM_UNICHAR WM_SYSDEADCHAR WM_SYSCHAR WM_SYSKEYUP
WM_SYSKEYDOWN WM_DEADCHAR WM_CHAR WM_KEYUP WM_KEYDOWN WM_KEYFIRST WM_INPUT
BM_SETDONTCLICK BM_SETIMAGE BM_GETIMAGE BM_CLICK BM_SETSTYLE BM_SETSTATE
BM_GETSTATE BM_SETCHECK BM_GETCHECK SBM_GETSCROLLBARINFO SBM_GETSCROLLINFO
SBM_SETSCROLLINFO SBM_SETRANGEREDRAW SBM_ENABLE_ARROWS SBM_GETRANGE SBM_SETRANGE
SBM_GETPOS SBM_SETPOS EM_GETIMESTATUS EM_SETIMESTATUS EM_CHARFROMPOS EM_POSFROMCHAR
EM_GETLIMITTEXT EM_GETMARGINS EM_SETMARGINS EM_GETPASSWORDCHAR EM_GETWORDBREAKPROC
EM_SETWORDBREAKPROC EM_SETREADONLY EM_GETFIRSTVISIBLELINE EM_EMPTYUNDOBUFFER
EM_SETPASSWORDCHAR EM_SETTABSTOPS EM_SETWORDBREAK EM_LINEFROMCHAR EM_FMTLINES
EM_UNDO EM_CANUNDO EM_SETLIMITTEXT EM_LIMITTEXT EM_GETLINE EM_SETFONT EM_REPLACESEL
EM_LINELENGTH EM_GETTHUMB EM_GETHANDLE EM_SETHANDLE EM_LINEINDEX EM_GETLINECOUNT
EM_SETMODIFY EM_GETMODIFY EM_SCROLLCARET EM_LINESCROLL EM_SCROLL EM_SETRECTNP
EM_SETRECT EM_GETRECT EM_SETSEL EM_GETSEL WM_NCXBUTTONDBLCLK WM_NCXBUTTONUP
WM_NCXBUTTONDOWN WM_NCMBUTTONDBLCLK WM_NCMBUTTONUP WM_NCMBUTTONDOWN WM_NCRBUTTONDBLCLK
WM_NCRBUTTONUP WM_NCRBUTTONDOWN WM_NCLBUTTONDBLCLK WM_NCLBUTTONUP WM_NCLBUTTONDOWN
WM_NCMOUSEMOVE WM_SYNCPAINT WM_GETDLGCODE WM_NCACTIVATE WM_NCPAINT WM_NCHITTEST
WM_NCCALCSIZE WM_NCDESTROY WM_NCCREATE WM_SETICON WM_GETICON WM_DISPLAYCHANGE
WM_STYLECHANGED WM_STYLECHANGING WM_CONTEXTMENU WM_NOTIFYFORMAT WM_USERCHANGED
WM_HELP WM_TCARD WM_INPUTLANGCHANGE WM_INPUTLANGCHANGEREQUEST WM_NOTIFY
WM_CANCELJOURNAL WM_COPYDATA WM_COPYGLOBALDATA WM_POWER WM_WINDOWPOSCHANGED
WM_WINDOWPOSCHANGING WM_COMMNOTIFY WM_COMPACTING WM_GETOBJECT WM_COMPAREITEM
WM_QUERYDRAGICON WM_GETHOTKEY WM_SETHOTKEY WM_GETFONT WM_SETFONT WM_CHARTOITEM
WM_VKEYTOITEM WM_DELETEITEM WM_MEASUREITEM WM_DRAWITEM WM_SPOOLERSTATUS
WM_NEXTDLGCTL WM_ICONERASEBKGND WM_PAINTICON WM_GETMINMAXINFO WM_QUEUESYNC
WM_CHILDACTIVATE WM_MOUSEACTIVATE WM_SETCURSOR WM_CANCELMODE WM_TIMECHANGE
WM_FONTCHANGE WM_ACTIVATEAPP WM_DEVMODECHANGE WM_WININICHANGE WM_CTLCOLOR
WM_SHOWWINDOW WM_ENDSESSION WM_SYSCOLORCHANGE WM_ERASEBKGND WM_QUERYOPEN
```

```
WM_QUIT WM_QUERYENDSESSION WM_CLOSE WM_PAINT WM_GETTEXTLENGTH WM_GETTEXT
WM_SETTEXT WM_SETREDRAW WM_ENABLE WM_KILLFOCUS WM_SETFOCUS WM_ACTIVATE
WM_SIZE WM_MOVE WM_DESTROY WM_CREATE WM_NULL SRCCOPY DIB_RGB_COLORS BI_RGB
->bmiColors ->bmiHeader BITMAPINFO ->biClrImportant ->biClrUsed ->biYPelsPerMeter
->biXPelsPerMeter ->biSizeImage ->biCompression ->biBitCount ->biPlanes
->biHeight ->biWidth ->biSize BITMAPINFOHEADER ->rgbReserved ->rgbRed ->rgbGreen
->rgbBlue RGBQUAD StretchDIBits DC_PEN DC_BRUSH DEFAULT_GUI_FONT SYSTEM_FIXED_FONT
DEFAULT_PALETTE DEVICE_DEFAULT_PALETTE SYSTEM_FONT ANSI_VAR_FONT ANSI_FIXED_FONT
OEM_FIXED_FONT BLACK_PEN WHITE_PEN NULL_BRUSH BLACK_BRUSH DKGRAY_BRUSH
GRAY_BRUSH LTGRAY_BRUSH WHITE_BRUSH GetStockObject COLOR_WINDOW RGB CreateSolidBrush
DeleteObject Gdi32 dpi-aware SetThreadDpiAwarenessContext VK_ALT GET_X_LPARAM
GET_Y_LPARAM IDI_INFORMATION IDI_ERROR IDI_WARNING IDI_SHIELD IDI_WINLOGO
IDI_ASTERISK IDI_EXCLAMATION IDI_QUESTION IDI_HAND IDI_APPLICATION LoadIconA
IDC_HELP IDC_APPSTARTING IDC_HAND IDC_NO IDC_SIZEALL IDC_SIZENS IDC_SIZEWE
IDC_SIZENESW IDC_SIZENWSE IDC_ICON IDC_SIZE IDC_UPARROW IDC_CROSS IDC_WAIT
IDC_IBEAM IDC_ARROW LoadCursorA PostQuitMessage FillRect ->rgbReserved
->fIncUpdate ->fRestore ->rcPaint ->fErase ->hdc PAINTSTRUCT EndPaint BeginPaint
GetDC PM_NOYIELD PM_REMOVE PM_NOREMOVE ->lPrivate ->pt ->time ->lParam
->wParam ->message ->hwnd MSG DispatchMessageA TranslateMessage PeekMessageA
GetMessageA ->bottom ->right ->top ->left RECT ->y ->x POINT CW_USEDEFAULT
IDI_MAIN_ICON DefaultInstance WS_TILEDWINDOW WS_POPUPWINDOW WS_OVERLAPPEDWINDOW
WS_CAPTION WS_TILED WS_ICONIC WS_CHILDWINDOW WS_GROUP WS_TABSTOP WS_POPUP
WS_CHILD WS_MINIMIZE WS_VISIBLE WS_DISABLED WS_CLIPSIBLINGS WS_CLIPCHILDREN
WS_MAXIMIZE WS_BORDER WS_DLGFRAME WS_VSCROLL WS_HSCROLL WS_SYSMENU WS_THICKFRAME
WS_MINIMIZEBOX WS_MAXIMIZEBOX WS_OVERLAPPED CreateWindowExA callback DefWindowProcA
SetForegroundWindow SW_SHOWMAXIMIZED SW_SHOWNORMAL SW_FORCEMINIMIZE SW_SHOWDEFAULT
SW_RESTORE SW_SHOWNA SW_SHWOMINNOACTIVE SW_MINIMIZE SW_SHOW SW_SHOWNOACTIVATE
SW_MAXIMIZED SW_SHOWMINIMIZED SW_NORMAL SW_HIDE ShowWindow ->lpszClassName
->lpszMenuName ->hbrBackground ->hCursor ->hIcon ->hInstance ->cbWndExtra
->cbClsExtra ->lpfnWndProc ->style WINDCLASSA RegisterClassA MB_CANCELTRYCONTINUE
MB_RETRYCANCEL MB_YESNO MB_YESNOCANCEL MB_ABORTRETRYIGNORE MB_OKCANCEL
MB_OK MessageBoxA User32 process-heap HeapReAlloc HeapFree HeapAlloc GetProcessHeap
win-key win-key? raw-key win-type init-console console-mode stderr stdout
stdin console-started FlushConsoleInputBuffer SetConsoleMode GetConsoleMode
GetStdHandle ExitProcess AllocConsole ENABLE_LVB_GRID_WORLDWIDE
DISABLE_NEWLINE_AUTO_RETURN
ENABLE_VIRTUAL_TERMINAL_PROCESSING ENABLE_WRAP_AT_EOL_OUTPUT ENABLE_PROCESSED_OUTPUT
ENABLE_VIRTUAL_TERMINAL_INPUT ENABLE_QUICK_EDIT_MODE ENABLE_INSERT_MODE
ENABLE_MOUSE_INPUT ENABLE_WINDOW_INPUT ENABLE_ECHO_INPUT ENABLE_LINE_INPUT
ENABLE_PROCESSED_INPUT STD_ERROR_HANDLE STD_OUTPUT_HANDLE STD_INPUT_HANDLE
invalid?ior d0<ior 0=ior ior FILE_END FILE_CURRENT FILE_BEGIN FILE_ATTRIBUTE_NORMAL
OPEN_EXISTING CREATE_ALWAYS FILE_SHARE_WRITE FILE_SHARE_READ GetFileSize
SetEndOfFile SetFilePointer MoveFileA DeleteFileA FlushFileBuffers CloseHandle
WriteFile ReadFile CreateFileA NULL wargs-convert wz>sz wargv wargc CommandLineToArgvW
Shell32 GetModuleHandleA GetCommandLineW GetLastError WaitForSingleObject
GetTickCount Sleep ExitProcess Kernel32 contains? dll sofunc GetProcAddress
LoadLibraryA WindowProcShim SetupCtrlBreakHandler boot_extra windows-builtins
calls
```

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

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
```

ENABLE_INSERT_MODE -- n

Constant, value: $0020

ENABLE_PROCESSED_INPUT -- n

Constant, value: $0001

win-type addr len --

Dispaly string on windows console

WM_CHAR -- 258

stack 258.

WM_CREATE -- 1

stack 1.

WM_DEADCHAR -- 259

stack 259.

WM_SYSDEADCHAR -- 258

stack 263.