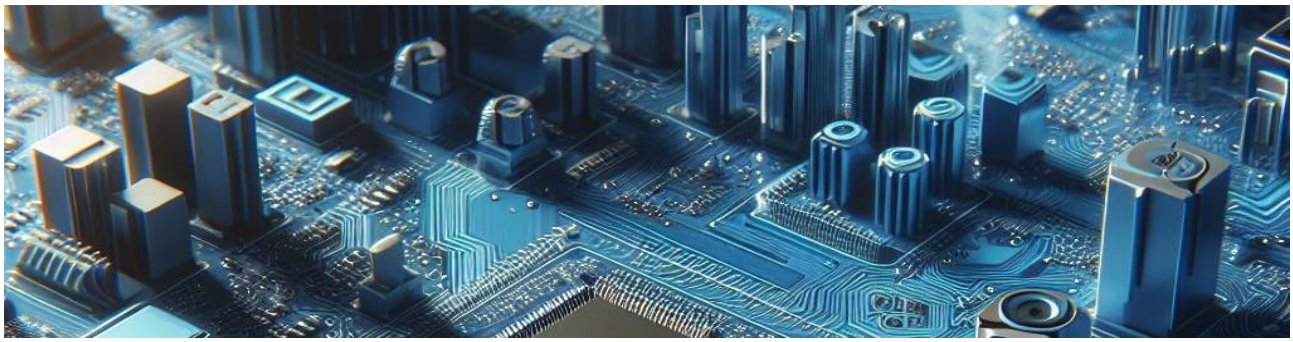
The great book
for eFORTH Windows

version 1.1 - 13 novembre 2024



Author

- Marc PETREMANN

Contents

Introduction

Je gère depuis 2019 plusieurs sites web consacrés aux développements en langage FORTH pour les cartes ARDUINO et ESP32, ainsi que les versions eForth web – Linux - Windows :

- ARDUINO : https://arduino-forth.com/
- ESP32 : https://esp32.arduino-forth.com/
- eForth web : https://eforth.arduino-forth.com/
- eForth Windows : https://eforthwin.arduino-forth.com/

Ces sites sont disponibles en deux langues, français et anglais. Chaque année je paie l'hébergement du site principal **arduino-forth.com**.

Il arrivera tôt ou tard – et le plus tard possible – que je ne sois plus en mesure d'assurer la pérennité de ces sites. La conséquence sera que les informations diffusées par ces sites disparaissent.

Ce livre est la compilation du contenu de mes sites web. Il est diffusé librement depuis un dépôt Github. Cette méthode de diffusion permettra une plus grande pérennité que des sites web.

Accessoirement, si certains lecteurs de ces pages souhaitent apporter leur contribution, ils sont bienvenus :

- pour proposer des chapitres ;
- pour signaler des erreurs ou suggérer des modifications ;
- pour aider à la traduction…

Aide à la traduction

Google Translate permet de traduire des textes facilement, mais avec des erreurs. Je demande donc de l'aide pour corriger les traductions.

En pratique, je fournis, les chapitres déjà traduits, dans le format LibreOffice. Si vous voulez apporter votre aide à ces traductions, votre rôle consistera simplement à corriger et renvoyer ces traductions.

La correction d'un chapitre demande peu de temps, de une à quelques heures.

Pour me contacter : petremann@arduino-forth.com

Installation on Windows

You can find the latest versions of eFORTH for WINDOWS here:

https://eforth.appspot.com/windows.html

The program version to be downloaded is in STABLE RELEASE or Beta Release.

Since µEforth version 7.0.7.21 only the 64 version remains available.

The downloaded program is directly executable. Once the program is downloaded, start by copying it to a working folder. Here, I chose to put the downloaded program in a folder named **eforth** .

To run µEforth Windows, click on the downloaded program and copy it to this eforth folder. If Windows issues a warning message:

- Click on Additional Information
- then click *Run Anyway*

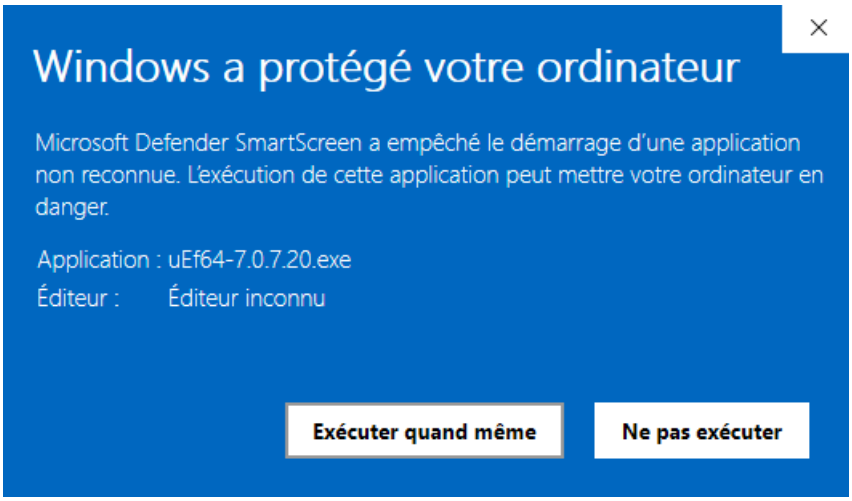


Figure 1: skip windows alert message

Once you have validated this choice, you will be able to run eForth like any other Windows program.

Setting up eForth Windows

eFORTH does not need to be installed to work. We simply run the downloaded file, here **uEf64-7.0.7.21.exe** . Here is the µEforth window:

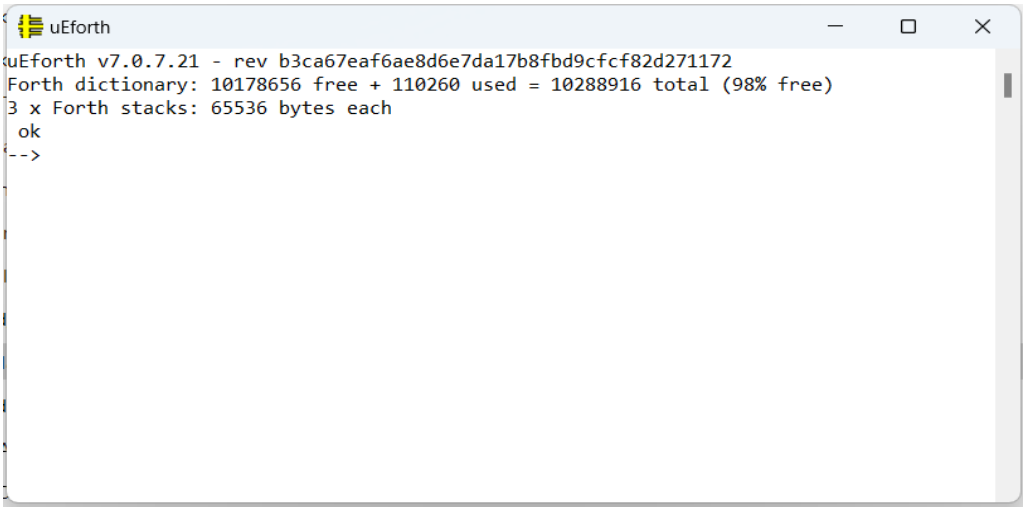


Figure 2: The µEforth window on Windows

To test that eForth is working properly, type **`words`** .

To exit eForth, type **`bye`** .

When eForth is open, you can create a shortcut to pin to the taskbar, which will make it easier to launch eForth again.

To change the background and text colors of eForth, hover over the µeForth logo, right-click and select *Properties:*

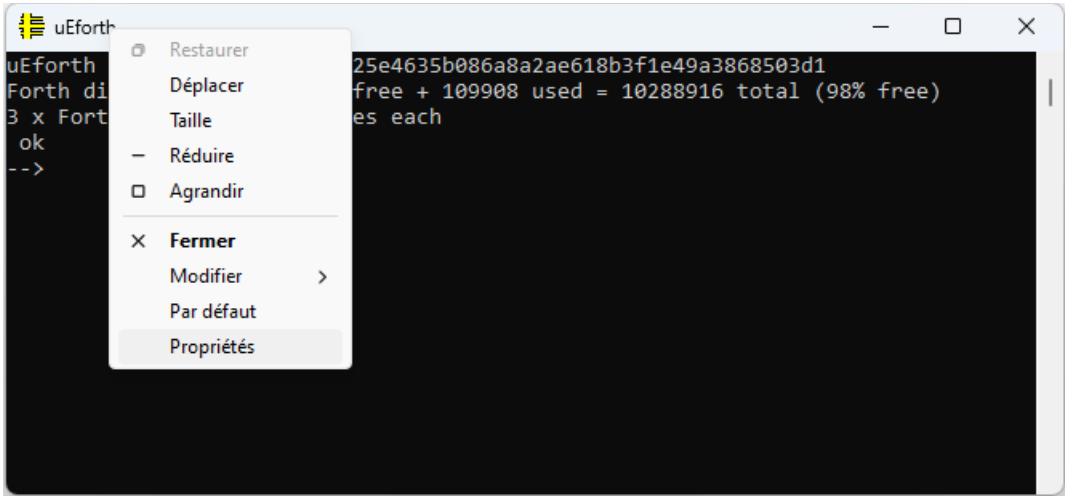


Figure 3: Selecting display properties

In Properties, click the *Colors tab* :

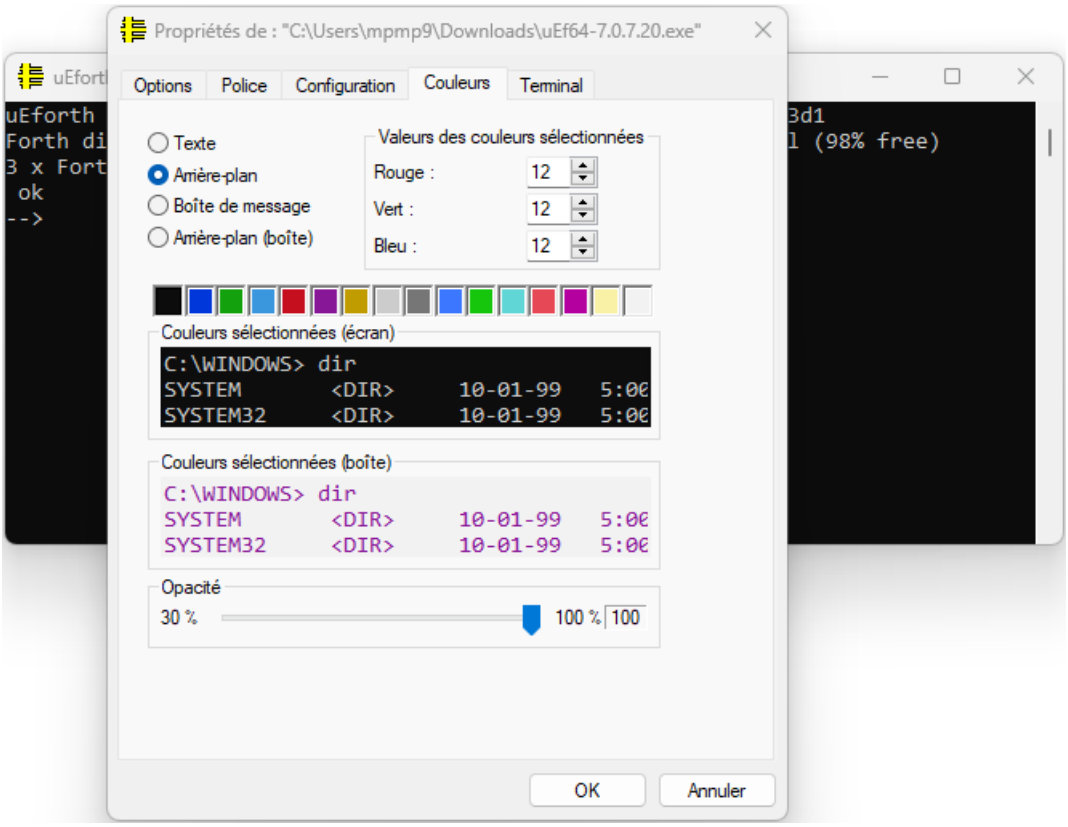


Figure 4: Choice of display colors

For my part, I chose to display the text in black on a white background. Click OK to validate this choice. The next time you launch eForth, you will find the colors selected as default settings for display in the eForth window.

Comments and program development

There is no IDE [1]to manage and present code written in FORTH language in a structured way. At worst, you use an ASCII text editor, at best a real IDE and text files:

- **edit** or **wordpad** on windows
- **PsPad** on windows
- **Netbeans** on Windows...

Here is a code snippet that could be written by a beginner:

```
: inGrid? { n gridPos -- fl } 0 { fl } gridPos getGridAddr for aft
getNumber n = if -1 to fl then then next drop  fl ;
```

This code will be perfectly compiled by eForth Windows. But will it remain understandable in the future if it needs to be modified or reused in another application?

Write readable FORTH code

Let's start with the naming of the word to be defined, here `inGrid?` . Eforth Windows allows you to write very long word names. The size of the defined words has no influence on the performance of the final application. We therefore have a certain freedom to write these words:

- in the manner of object programming in javaScript: `grid.test.number`
- in the CamelCoding way `gridTestNumber`
- for programmers wanting very understandable code `is-number-in-the-grid`
- programmer who likes concise code `gtn?`

There is no rule. The main thing is that you can easily proofread your FORTH code. However, FORTH language computer programmers have certain habits:

- `LOTTO_NUMBERS_IN_GRID` uppercase constants
- definition word of other words `lottoNumber:` word followed by a colon;
- address transformation word `>date`, here the address parameter is incremented by a certain value to point to the appropriate data;
- memory storage word `date@` or `date!`
- Data display word `.date`

1 Integrated Development Environment

And what about naming FORTH words in a language other than English? Again, there is only one rule: **total freedom** ! Be careful though, eForth Windows does not accept names written in alphabets other than the Latin alphabet. You can, however, use these alphabets for comments:

```
: .date     \ Плакат сегодняшней даты
    ….code…   ;
```

Or

```
: .date     \ 海報今天的日期
    ….code…   ;
```

Source code indentation

Whether the code is on two lines, ten lines or more, it has no effect on the performance of the code once compiled. So, you might as well indent your code in a structured way:

- one line per word of control structure **`if else then`** , **`begin while repeat…`** For the word if, we can precede it with the logical test that it will process;
- one line per execution of a predefined word, preceded where appropriate by the parameters of this word.

Example :

```
: inGrid? { n gridPos -- fl }
    0 { fl }
    gridPos getGridAddr
    for
        aft
            getNumber n =
            if
                -1 to fl
            then
        then
    next
    drop
    fl
  ;
```

If the code processed in a control structure is sparse, the FORTH code can be compacted:

```
: inGrid? { n gridPos -- fl }
    0 { fl }   gridPos getGridAddr
    for aft
            getNumber n =
            if -1 to fl  then
        then
    next
    drop  fl
  ;
```

This is often the case with `case of endof endcase` structures ;

```
: socketError ( -- )
    errno dup
    case
        2 of      ." No such file "                    endof
        5 of      ." I/O error "                       endof
        9 of      ." Bad file number "                 endof
       22 of      ." Invalid argument "                endof
    endcase
    . quit
  ;
```

Comments

Like any programming language, the FORTH language allows the addition of comments in the source code. Adding comments has no impact on the performance of the application after compiling the source code.

In FORTH language, we have two words to delimit comments:

- the word `(` followed by at least one space character. This comment is completed by the character `)` ;
- the word `\` followed by at least one space character. This word is followed by a comment of any size between this word and the end of the line.

The word `(` is widely used for stack comments. Examples:

```
dup   ( n – n n )
swap  ( n1 n2 – n2 n1 )
drop  ( n -- )
emit  ( c -- )
```

Stack Comments

As we have just seen, they are marked by `(` and `)` . Their content has no effect on the FORTH code during compilation or execution. We can therefore put anything between `(` and `)` . As for stack comments, we will remain very concise. The `-- sign` symbolizes the action of a FORTH word. The indications appearing before `--` correspond to the data placed on the data stack before the execution of the word. The indications appearing after `--` correspond to the data left on the data stack after the execution of the word. Examples:

- `words ( -- )` means that this word does not process any data on the data stack;
- `emit ( c -- )` means that this word processes an input data and leaves nothing on the data stack;
- `bl (--32)` means that this word does not process any input data and leaves the decimal value 32 on the data stack;

There is no limitation on the amount of data processed before or after the word is executed. As a reminder, the indications between `(` and `)` are for information purposes only.

Meaning of stack parameters in comments

To begin, a very important little clarification is necessary. This concerns the size of the data in the stack. With eForth Windows, the stack data takes up 8 bytes. These are therefore integers in 64-bit format. So what do we put on the data stack? With eForth Windows, it will **ALWAYS be 64-BIT DATA** ! An example with the word `c` !:

```
create myDelemiter
    0 c,
64 myDelimiter c!    ( c addr -- )
```

`c` parameter indicates that we are stacking an integer value in 64-bit format, but whose value will always be included in the interval [0..255].

The standard parameter is always `n` . If there are several integers, we will number them: `n1 n2 n3` , etc.

So we could have written the previous example like this:

```
create myDelemiter
    0 c,
64 myDelimiter c!    ( n1 n2 -- )
```

But it is much less explicit than the previous version. Here are some symbols that you will see throughout the source codes:

- **addr** indicates a literal memory address or one delivered by a variable;
- **c** indicates an 8-bit value in the range [0..255]
- **d** indicates a double precision value.
  Not used with eForth Windows which is already 32 or 64 bit;
- **fl** indicates a boolean value, 0 or non-zero;
- **n** indicates an integer. 32- or 64-bit signed integer for eForth Windows;
- **str** indicates a string. Equivalent to `addr len --`
- **u** indicates an unsigned integer

There is nothing to prevent us from being a little more explicit:

```
: SQUARE ( n -- n-exp2 )
    dup *
  ;
```

Word Definition Words Comments

Definition words use `create` and `does>` . For these words, it is recommended to write stack comments like this:

```
\ define a command or data stream for SSD1306
: streamCreate: ( comp: <name> | exec: -- addr len )
    create
        here    \ leave current dictionnary pointer on stack
        0 c,    \ initial lenght data is 0
    does>
        dup 1+ swap c@
        \ send a data array to SSD1306 connected via I2C bus
        sendDatasToSSD1306
  ;
```

Here the comment is split into two parts by the `| character` :

- on the left, the action part when the definition word is executed, prefixed by `comp:`
- on the right the action part of the word that will be defined, prefixed by `exec:`

At the risk of insisting, this is not a standard. These are only recommendations.

Text comments

They are indicated by the word `\` followed by at least one space character and explanatory text:

```
\ store at <WORD> addr length of datas compiled beetween
\ <WORD> and here
: ;endStream ( addr-var len ---)
    dup 1+ here
    swap -      \ calculate cdata length
    \ store c in first byte of word defined by streamCreate:
    swap c!
  ;
```

These comments can be written in any alphabet supported by your source code editor:

```
\ 儲存在 <WORD> addr 之間編譯的資料長度
\ <WORD> 和這裡
: ;endStream ( addr-var len ---)
    dup 1+ here
    swap -      \ 計算 cdata 長度
    \ 將 c 儲存在由 StreamCreate 定義的字的第一個位元組中：
    swap c!
  ;
```

Comment at the beginning of the source code

With intensive programming practice, you quickly end up with hundreds, even thousands of source files. To avoid file selection errors, it is strongly recommended to mark the beginning of each source file with a comment:

```
\ ******************************************************************
\ key word for UT8 characters
\    Filename:       uekey.fs
\    Date:           29 nov 2023
\    Updated:        29 nov 2023
\    File Version:   1.1
\    MCU:            Linux / Web / Windows
\    Forth:          eForth Windows
\    Copyright:      Marc PETREMANN
\    Author:         Marc PETREMANN
\    GNU General Public License
\ ******************************************************************
```

All this information is at your discretion. It can become very useful when you come back to the contents of a file months or years later.

Finally, do not hesitate to comment and indent your source files in FORTH language.

Diagnostic and tuning tools

The first tool concerns the compilation or interpretation alert:

```
3 5 25 --> : TEST ( ---)
 ok
3 5 25 -->     [ HEX ] ASCII A DDUP    \ DDUP don't exist
```

Here, the word **DDUP** does not exist. Any compilation after this error will fail.

The decompiler

In a conventional compiler, the source code is transformed into executable code containing the reference addresses to a library equipping the compiler. To have executable code, the object code must be linked. At no time can the programmer access the executable code contained in his libraries with the compiler's resources alone.

With eForth Windows, the developer can decompile his definitions. To decompile a word, simply type **see** followed by the word to decompile:

```
: C>F ( øC --- øF) \ Conversion Celsius in Fahrenheit
    9 5 */ 32 +
  ;
see c>f
\ display:
: C>F
    9 5 */ 32 +
```

```
;
```

Many words in the eForth Windows FORTH dictionary can be decompiled.

Decompiling your words allows you to detect possible compilation errors.

Memory dump

Sometimes it is desirable to be able to see the values that are in memory. The **dump word** accepts two parameters: the starting address in memory and the number of bytes to view:

```
create myDATAS 01 c, 02 c, 03 c, 04 c,
hex
myDATAS 4 dump      \ displays :
3FFEE4EC                                                01 02 03 04
```

Stack monitor

The contents of the data stack can be displayed at any time using the **.s** keyword . Here is the definition of the **.DEBUG** keyword which uses **.s** :

```
variable debugStack

: debugOn ( -- )
    -1 debugStack !
  ;

: debugOff ( -- )
    0 debugStack !
  ;

: .DEBUG
    debugStack @
    if
        cr ." STACK: " .s
        key drop
    then
  ;
```

To exploit **.DEBUG** , simply insert it in a strategic location in the word to be debugged:

```
\ example of use:
: myTEST
    128 32 do
        i   .DEBUG
        emit
    loop
  ;
```

Here we will display the contents of the data stack after executing the word i in our **do loop** . We activate the debug and execute **myTEST** :

```
debugOn
myTest
\ displays:
\ STACK: <1> 32
\ 2
\ STACK: <1> 33
\ 3
\ STACK: <1> 34
\ 4
\ STACK: <1> 35
\ 5
\ STACK: <1> 36
\ 6
\ STACK: <1> 37
\ 7
\ STACK: <1> 38
```

When debugging is enabled by **`debugOn`**, each display of the contents of the data stack pauses our **`do loop`**. Run **`debugOff`** to make the word **`myTEST`** run normally.

Dictionary / Stack / Variables / Constants

Expand the dictionary

Forth belongs to the class of woven interpreter languages. This means that it can interpret commands typed on the console, as well as compile new subroutines and programs.

The Forth compiler is part of the language and special words are used to create new dictionary entries (i.e. words). The most important ones are `:` (start a new definition) and `;` (completes the definition). Let's try this by typing:

```
: *+ * + ;
```

What happened? The action of : is to create a new dictionary entry named `*+` and switch from interpreter mode to compile mode. In compile mode, the interpreter looks up words and, rather than executing them, installs pointers to their code. If the text is a number, instead of pushing it onto the stack, eForth Windows builds the number in the dictionary in the space allocated for the new word, following special code that pushes the stored number onto the stack each time the word is executed. The action of executing `*+` is therefore to sequentially execute the previously defined words `*` and `+` .

The word `;` is special. It is an immediate word and is always executed, even if the system is in compile mode. What `;` does is twofold. First, it installs code that returns control to the next external level of the interpreter, and second, it returns from compile mode to interpret mode.

Now try your new word:

```
decimal 5 6 7 *+ . \ displays 47
```

This example illustrates two main working activities in Forth: adding a new word to the dictionary, and trying it out once it has been defined.

Stacks and Reverse Polish Notation

Forth has an explicitly visible stack that is used to pass numbers between words (commands). Using Forth effectively requires you to think in terms of a stack. This can be difficult at first, but as with anything, it gets much easier with practice.

In FORTH, the stack is analogous to a stack of cards with numbers written on them. Numbers are always added to the top of the stack and removed from the top of the stack. eForth Windows integrates two stacks: the parameter stack and the return stack, each consisting of a number of cells that can hold 16-bit numbers.

The FORTH input line:

```
decimal 2 5 73 -16
```

leaves the parameter stack as is

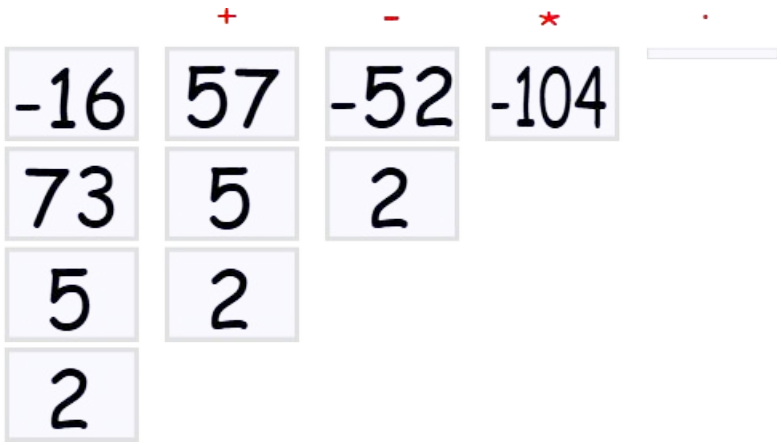
Cell	content	comment
0	-16	(TOS) Top stack
1	73	(NOS) Next in the stack
2	5	
3	2	

We will typically use zero-based relative numbering in Forth data structures such as stacks, arrays, and tables. Note that when a sequence of numbers is entered like this, the rightmost number becomes *TOS* and the leftmost number is at the bottom of the stack.

Suppose we follow the original input line with the line

```
+ - * .
```

The operations would produce the successive stack operations:



After the two lines, the console displays:

```
decimal 2 5 73 -16      \ displays: 2 5 73 -16 ok
+ - * .                 \ displays: -104 ok
```

Note that eForth Windows conveniently displays the stack elements as it interprets each line, and the value of -16 is displayed as a 32-bit unsigned integer. Also, the word . consumes the data value -104, leaving the stack empty. If we execute . on the now empty stack, the external interpreter aborts with a stack pointer error STACK UNDERFLOW ERROR.

The programming notation where the operands appear first, followed by the operator(s) is called Reverse Polish Notation (RPN).

Handling the parameter stack

Being a stack-based system, eForth Windows must provide ways to put numbers on the stack, to remove them, and to rearrange their order. We have already seen that we can put numbers on the stack simply by typing them. We can also integrate the numbers into the definition of a FORTH word.

The word `drop` removes a number from the top of the stack, putting the next number on top. The word `swap` swaps the first 2 numbers. `dup` copies the number on top, pushing all the other numbers down. `rot` rotates the first 3 numbers. These actions are shown below.

The Return Stack and Its Uses

When compiling a new word, eForth Windows establishes links between the calling word and previously defined words that are to be invoked by the execution of the new word. This linking mechanism, at runtime, uses the return stack (rstack). The address of the next word to be invoked is placed on the return stack so that when the current word is completed during execution, the system knows where to jump to the next word. Since words can be nested, there must be a stack of these return addresses.

In addition to serving as a reservoir of return addresses, the user can also store and retrieve from the return stack, but this must be done carefully because the return stack is essential to program execution. If you use the return stack for temporary storage, you must return it to its original state, otherwise you will likely crash the eForth Windows system. Despite the danger, there are times when using the return stack as temporary storage can make your code less complex.

To store on the stack, use `>r` to move the top of the parameter stack to the top of the return stack. To retrieve a value, `r>` moves the top value of the return stack to the top of the parameter stack. To simply remove a value from the top of the stack, there is the word `rdrop` . The word `r@` copies the top of the return stack to the parameter stack.

Memory Usage

In eForth Windows, 32-bit numbers are fetched from memory to the stack by the word `@` (fetch) and stored from the top to memory by the word `!` (store). `@` expects an address on the stack and replaces the address with its contents. `!` expects a number and an address to store it. It places the number in the memory location referenced by the address, consuming both parameters in the process.

Unsigned numbers that represent 8-bit (byte) values can be placed in character-sized memory cells using `c@` and `c!` .

```
create testVar
cell allot
$f7 testVar c!
testVar c@ .        \ displays 247
```

Variables

A variable is a named location in memory that can store a number, such as the intermediate result of a calculation, off the stack. For example:

```
variable x
```

creates a storage location named, `x` , which executes by leaving the address of its storage location on top of the stack:

```
x .          \ displays the address
```

We can then collect or store at this address:

```
variable x
3 x !
x @ .        \ displays: 3
```

Constants

A constant is a number that you would not want to change while a program is running. The result of executing the word associated with a constant is the value of the data remaining on the stack.

```
\ defines extrem values for alpha channel
255 constant SDL_ALPHA_OPAQUE
0   constant SDL_ALPHA_TRANSPARENT
```

Pseudo-constant values

A value defined with value is a hybrid type of variable and constant. We define and initialize a value and it is invoked as we would a constant. We can also change a value as we can change a variable.

```
decimal
```

```
13 value thirteen
thirteen .            \ display: 13
47 to thirteen
thirteen .            \ display: 47
```

The word `to` also works in word definitions, replacing the value following it with whatever is currently on top of the stack. You have to be careful that `to` is followed by a value defined by `value` and not something else.

Basic tools for memory allocation

The words `create` and `allot` are the basic tools for reserving memory space and attaching a label to it. For example, the following transcription shows a new graphic-array dictionary entry :

```
create graphic-array ( --- addr )
%00000000 c,
%00000010 c,
%00000100 c,
%00001000 c,
%00010000 c,
%00100000 c,
%01000000 c,
%10000000 c,
```

When executed, the `graphic-array` word will push the address of the first entry.

We can now access the memory allocated to `graphic-array` using the fetch and store words explained earlier. To calculate the address of the third byte allocated to `graphic-array` we can write `graphic-array 2 +` , remembering that indices start at 0.

```
30 graphic-array 2 + c!
graphic-array 2 + c@ .          \ displays 30
```

Version v 7.0.7.15

FORTH

-	-1	-rot	,	;	:	:noname	!
?	?do	?dup	.	."	.s	'	(local)
[	[']	[char]	[ELSE]	[IF]	[THEN]	]	{
}transfer	@	*	*/	*/MOD	/	/mod	#
#!	#>	#fs	#s	#tib	+	+!	+loop
+to	<	<#	<=	<>	=	>	>=
>BODY	>flags	>flags&	>in	>link	>link&	>name	>params
>R	>size	0	0<	0<>	0=	1	1-
1/F	1+	10	2!	2@	2*	2/	2drop
2dup	3dup	4*	4/	41	abort	abort"	abs
accept	afliteral	aft	again	ahead	align	aligned	allocate
allot	also	AND	ansi	argc	argv	ARSHIFT	asm
assert	at-xy	base	begin	bg	BIN	binary	bl
blank	block	block-fid	block-id	buffer	bye	c,	C!
C@	CASE	cat	catch	CELL	cell/	cell+	cells
char	CLOSE-FILE	cmove	cmove>	CONSTANT	context	copy	cp
cr	CREATE	CREATE-FILE	current	decimal	default-key	default-key?	
default-type		default-use	defer	DEFINED?	definitions	DELETE-FILE	depth
do	DOES>	DROP	dump	dump-file	DUP	echo	editor
else	emit	empty-buffers		ENDCASE	ENDOF	erase	evaluate
EXECUTE	exit	extract	F-	f.	f.s	F*	F**
F/	F+	F<	F<=	F<>	F=	F>	F>=
F>S	F0<	F0=	FABS	FATAN2	fconstant	FCOS	fdepth
FDROP	FDUP	FEXP	fg	file-exists?		FILE-POSITION	
FILE-SIZE	fill	FIND	fliteral	FLN	FLOOR	flush	FLUSH-FILE
FMAX	FMIN	FNEGATE	FNIP	for	forget	FORTH	forth-builtins
FOVER	FP!	FP@	fp0	free	FROT	FSIN	FSINCOS
FSQRT	FSWAP	fvariable	graphics	here	hex	hld	hold
I	if	IMMEDIATE	include	included	included?	internals	invert
is	J	K	key	key?	L!	latestxt	leave
list	literal	load	loop	LSHIFT	max	min *	mod
ms	ms-ticks	mv	n.	needs	negate	next	nip
nl	NON-BLOCK	normal	octal	OF	ok	only	open-blocks
OPEN-FILE	OR	order	OVER	pad	page	PARSE	pause
pause?	PI	postpone	postpone,	precision	previous	prompt	quit
r"	R@	R/O	R/W	R>	r|	r~	rdrop
READ-FILE	recognizers	recurse	refill	remaining	remember	RENAME-FILE	repeat
REPOSITION-FILE		required	reset	resize	RESIZE-FILE	restore	revive
rm	rot	RP!	RP@	rp0	RSHIFT	s"	S>F
s>z	save	save-buffers		scr	sealed	see	set-precision
set-title	sf,	SF!	SF@	SFLOAT	SFLOAT+	SFLOATS	sign
SL@	SP!	SP@	sp0	space	spaces	start-task	startswith?
startup:	state	str	str=	streams	structures	SW@	SWAP
task	tasks	terminate	then	throw	thru	tib	to
touch	transfer	transfer{	type	u.	U/MOD	U<	UL@
UNLOOP	until	update	use	used	UW@	value	VARIABLE
visual	vlist	vocabulary	W!	W/O	while	windows	words

```
WRITE-FILE  XOR         z"          z>s
```

windows

```
process-heap HeapReAlloc HeapFree HeapAlloc GetProcessHeap 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_WNT_CONVERTREQUESTEX WM_UNICHAR WM_SYSDEADCHAR WM_SYSCHAR
WM_SYSKEYUP WM_SYSKEYDOWN WM_DEADCHAR WM_CHAR WM_KEYUP WM_KEYDOWN 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 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 d0NULL wargs-convert wz>sz wargv
wargc CommandLineToArgvW Shell32 GetModuleHandleA GetCommandLineW GetLastError
WaitForSingleObject GetTickCount Sleep ExitProcess Kernel32 contains? dll
sofunc GetProcAddress LoadLibraryA WindowProcShim SetupCtrlBreakHandler
windows-builtins calls
```

Ressources

English

- **ESP32forth** page maintained by Brad NELSON, the creator of ESP32forth. You will find all versions there (ESP32, Windows, Web, Linux...)
  https://esp32forth.appspot.com/ESP32forth.html

French

- **eForth** two-language site (French, English) with lots of examples
  https://eforthwin.arduino-forth.com/

GitHub

- **Ueforth** resources maintained by Brad NELSON. Contains all Forth and C source files for ESP32forth and ueForth Windows, Linux and web.
  https://github.com/flagxor/ueforth
- **eForth Windows** source codes and documentation for eForth Windows. Resources maintained by Marc PETREMANN
  https://github.com/MPETREMANN11/eForth-Windows
- **eForth SDL2 project** for eForth Windows
  https://github.com/MPETREMANN11/SDL2-eForth-windows

Facebook

- **Eforth** group for eForth Windows
  https://www.facebook.com/groups/785868495783000

Index lexical