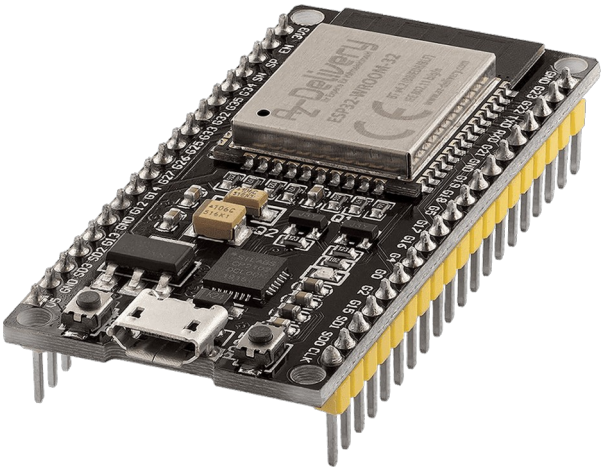
Word Glossary for ESP32forth v7.0.7.21

ESP32forth is a powerful forth tool set created by **Bradley D Nelson** and the late **Dr. Hanson Ting** for the low cost ESP32 module. This document is a word glossary for reference whilst programming. There are also some useful links to reference material at the back.

The headings and words in the following tables are in alphabetic order to speed up searches. Most document readers will show a 'Table of Contents' strip you can click on to jump to a particular section. Use ctrl F to search for a specific word. Immediate words (that run at compile time) are shown in red. The right-hand column indicates the ESP32forth vocabulary in which the word is located.

There are plenty of gaps and probably a few errors. Any contributions welcome via the Forth2020 forum on Facebook



ESP32 is a series of low-cost, low-power system on a chip microcontrollers with integrated Wi-Fi and dual-mode Bluetooth. The ESP32 series employs either a Tensilica Xtensa LX6 microprocessor in both dual-core and single-core variations, Xtensa LX7 dual-core microprocessor or a single-core RISC-V microprocessor and includes built-in antenna switches, RF balun, power amplifier, low-noise receive amplifier, filters, and power-management modules.

ANSI terminal control

The terminal operates at 115200 baud, 8 data bits, 1 stop bit and no parity.

at-xy	(x y –)	Set the cursor at x, y on the terminal screen. 0,0 being the top left corner.	forth
bel	(–)	Sound the terminal bell	ansi
bg	(n –)	Set the back ground colour to n	forth
clear-to-eol	(–)	Clear the line from the cursor position to the right margin	ansi
esc	(–)	sends the esc char to the terminal - many ANSI terminal commands start with the esc character	ansi
fg	(n –)	Set the character (or foreground) colour to n	forth
hide	(–)	Hide the cursor	ansi
normal	(–)	Return the terminal screen to black background, white	forth

		characters	
page	(–)	Scroll the terminal screen up enough so that a blank screen is shown	forth
scroll-down	(–)	Scroll the cursor down one line	ansi
scroll-up	(–)	Scroll the cursor up one line	ansi
set-title	(a n –)	Changes the text shown in the title bar of the terminal window to the string at a, n	forth
show	(–)	Make the cursor visible	ansi
terminal-restore	(–)	Scroll the terminal back down, restoring the terminal to the state when terminal-save was executed	ansi
terminal-save	(–)	Scroll the present terminal up off the screen, leaving a blank screen	ansi

Block File System

block	(n -- a)	Get a 1024 character block	forth
block-fid	(– n)	value, default = -1	forth
block-id	(-- n)	value, default = -1	forth
buffer	(n -- a)	Get a 1024 byte block without regard to old contents	forth
copy	(from to --)	Copy contents of block 'from' to block 'to'	forth
default-use		deferred word, defaults to common-default-use	forth
empty-buffers	(--)	Empty all buffers	forth
flush	(--)	Save and empty all buffers	forth
list	(n --)	List block n of 1024 characters to the display	forth
load	(n --)	Evaluate block n of 1024 characters as the input stream	forth
open-blocks	(a n --)	Open a file as the block file	forth
save-buffers	(--)	Save all buffers	forth
scr	(-- adr)	Pointer to last listed block	forth
thru	(a b --)	Load blocks a thru b	forth
update	(--)	Mark the last block modified	forth
use	("name" --)	Use "name" as the blockfile, e.g. USE /spiffs/foo	forth

Block File Editor

a	(n "text" --)	Add (insert) a line in the current block with the words that follow in the input stream, terminated with <cr>	editor
d	(n --)	Delete a line in the current block	editor
e	(n --)	Clear a line in the current block	editor
l	(–)	List the current 1024 character block	editor
n	(–)	Move to the next 1024 character block	editor
p	(–)	Move to the previous 1024 character block	editor
r	(n "text" --)	Replace a line in the current block with the words that follow in the input stream, terminated with <cr>	editor
wipe	(--)	Erase the 1024 character block	editor
editor	(--)	vocabulary name of the block file editor	forth

Branching

[ELSE]	(–)	Interpret time ELSE *	forth
[IF]	(f –)	Interpret time IF (conditional interpretation of words that follow, dependent on flag f) *	forth
[THEN]	(–)	Interpret time THEN *	forth
aft	(--)	: aft-example (n --) for <words that run only 1st interation> aft <words that run for all but the 1st iteration> then <words that run on all iterations> next ;	forth
ahead	(--)	continue execution after then e.g. : myword333 ahead 222 444 then ; running myword would leave 333 on the stack	forth
case	(–)	Mark the start of the CASE...OF...ENDOF...ENDCASE structure	forth
DEFINED?	("name" -- xt | 0)	If the name that follows in the input stream is found in the dictionary, the execution ad dress of the word is placed on the stack, else 0 if not found e.g. DEFINED? CREATE returns 1073637288. DEFINED? BABA returns 0	forth
else	(–)	part of a conditional structure e.g. if <forth words else <more forth words> then e.g. if <forth words> then	forth
endcase	(n1 –)	Mark the end of the CASE...OF.....ENDCASE structure	

endof	(–)	Mark the end of the OF...ENDOF part of the CASE structure	forth
if	(flag –)	Conditional structure, which executes depends on flag e.g. if <forth words else <more forth words> then e.g. if <forth words> then	forth
of	(n1 n2 – else -- n1)	If the two values on the stack are not equal, discard the top value and continue execution following the next ENDOF. Otherwise, discard both values and continue execution in line	forth
then	(--)	part of a conditional structure e.g. if <forth words else <more forth words> then e.g. if <forth words> then	forth

* [IF] [ELSE] [THEN] seem at the moment to be limited to appearing on the same line only – not multiline

CASE code example

```
: test          (  n – )
        case
                0 of ." zero" endof
                1 of ." one" endof
                2 of ." two" endof
                ." many"                        \ this code runs if none of the cases are met
        endcase ;
```

[IF] [THEN] example (n1 n2 --) Demonstrates creating words with no names, which can nevertheless be executed by execution token - saves space in the dictionary if the word is never used by name

```
\ Place this at the top of a forth source file

DEFINED? *codename* [IF] forget *codename* [THEN]
: *codename* ;

\ the string *codename* can be anything unique to suit the code file being loaded
\ if that word is already defined, then the compiled words are forgotten from the
\ dictionary before being recompiled
```

Camera

camera-server	(–)		forth
s->set_xclk	(s time xclk)		camera
s->set_pll	(s bypass mul sys root pre seld5 pclken pclk)		camera
s->set_res_raw	(s startX startY endX endY offsetX offsetY totalX totalY outputX outputY scale binning)		camera
s->set_reg	(s reg mask value)		camera
s->get_reg	(s reg mask)		camera
s->set_lenc	(s enable)		camera
s->set_raw_gma	(s enable)		camera
s->set_ae_level	(s level)		camera
s->set_wb_mode	(s mode)		camera
s->set_special_effect	(s effect)		camera
s->set_aec_value	(s gain)		camera
s->set_agc_gain	(s gain)		camera
s->set_awb_gain	(s enable)		camera
s->set_aec2	(s enable)		camera
s->set_vflip	(s enable)		camera
s->set_hmirror	(s enable)		camera
s->set_exposure_ctrl	(s enable)		camera
s->set_gain_ctrl	(s enable)		camera
s->set_whitebal	(s enable)		camera
s->set_colorbar	(s enable)		camera
s->set_quality	(s quality)		camera
s->set_gainceiling	(s gainceil)		camera
s->set_denoise	(s level)		camera
s->set_sharpness	(s level)		camera
s->set_saturation	(s level)		camera
s->set_brightness	(s level)		camera
s->set_contrast	(s level)		camera
s->set_framesize	(s framesize)		camera
s->set_pixformat	(s pixformat)		camera

s->reset	(s)		camera
s->init_status	(s)		camera
s->xclk_freq_hz	(a)		camera
fb->usec			camera
fb->sec			camera
fb->format			camera
fb->height			camera
fb->width			camera
fb->len			camera
fb->buf			camera
field@	(n -- n)		camera
camera-format		constant	camera
camera-frame-size		constant	camera
camera-jpeg-quality		constant	camera
camera-fb-count		constant	camera
camera-config			camera
FRAMESIZE_UXGA	(– 13)	constant	camera
FRAMESIZE_SXGA	(– 12)	constant	camera
FRAMESIZE_HD	(– 11)	constant	camera
FRAMESIZE_XGA	(– 10)	constant	camera
FRAMESIZE_SVGA	(– 9)	constant	camera
FRAMESIZE_VGA	(– 8)	constant	camera
FRAMESIZE_HVGA	(– 7)	constant	camera
FRAMESIZE_CIF	(– 6)	constant	camera
FRAMESIZE_QVGA	(– 5)	constant	camera
FRAMESIZE_240x240	(– 4)	constant	camera
FRAMESIZE_HQVGA	(– 3)	constant	camera
FRAMESIZE_QCIF	(– 2)	constant	camera
FRAMESIZE_QQVGA	(– 1)	constant	camera
FRAMESIZE_96x96	(– 0)	constant	camera
PIXFORMAT_RGB555	(– 7)	constant	camera
PIXFORMAT_RGB444	(– 6)	constant	camera
PIXFORMAT_RAW	(– 5)	constant	camera
PIXFORMAT_RGB888	(– 4)	constant	camera
PIXFORMAT_JPEG	(– 3)	constant	camera

PIXFORMAT_GRAYSCALE	(– 2)	constant	camera
PIXFORMAT_YUV422	(– 1)	constant	camera
PIXFORMAT_RGB565	(– 0)	constant	camera

Character I/O

#tib	(-- addr)	variable, Contains the size of the terminal input buffer (TIB)	forth
>in	(-- addr)	variable, contains address of a cell containing the offset in characters from the start of the input buffer to the start of the parse area	forth
."	(“string” –)	display the string that follows in the input stream until a terminating “	forth
accept	(b u1 -- u2)	accepts u1 characters to buffer b. u2 returned is the actual count of characters received. Terminates when cr entered or u1 chars received. Supports delete for input correction	forth
bl	(-- 32)	returns the value of the SPACE char	forth
char	("character" -- c)	Convert the non-space char that follows in the input stream and place it's value tos e.g. char 0 places 48 on top of the stack	forth
cr	(--)	send carriage return, line feed to the display	forth
emit	(c –)	display the ascii character c	forth
key	(– chr)	deferred word - reads the next character in the input stream - defaults to word serial-key	forth
key?	(-- cnt)	deferred word - returns cnt, the number of characters waiting to be read. cnt=0 if no characters waiting - defaults to word serial-key?	forth
nl	(-- 10)	the value of the NEWLINE character	forth
ok	(--)	display the string ok	forth
page	(--)	Emit 30 CR characters to the display	forth
PARSE	(c "wordtoparse" -- addr cnt)	Parse the next word in the input stream, terminating on character c. Leave the address and character count cnt of word.If the parse area was empty then cnt=0	forth
prompt		send ok and <cr> to the display	forth
space	(--)	Emit a space character to the display	forth
tib	(-- addr)	returns the address of the the terminal input buffer	forth

		where input text string is held.	
type	(addr n --)	deferred word - display an n long character string, located at addr, on the display - as a default executes serial-type	forth
ip.	(n --)	print n in IP address format e.g. 192.168.1.2	web-interface
ip#	(n1 -- n2)	print one section of an ip address	web-interface

Comment

(	("string"--)	Start of a 1 line comment, terminated with). Most often used to document the stack effect of word	forth
****		Start of a 1 line comment, no termination needed	forth

Comparison

<	(n1 n2 -- f)	f=true of n1 less than n2, else f=false	forth
<=	(n1 n2 -- f)	f=true if n1 less than or equal n2, else f=false	forth
<>	(n1 n2 -- f)	f=true if n1 not equal n2, else f=false	forth
=	(n1 n2 -- f)	f=true if n1 equals n2, else f=false	forth
>	(n1 n2 -- f)	f=true if n1 greater than n2, else f=false	forth
>=	(n1 n2 -- f)	f=true if n1 greater than or equal n2, else f=false	forth
0<	(n -- f)	f=true if n less than zero, else f=false	forth
0<>	(n -- f)	f=true if n not equal zero, else f=false	forth
0=	(n -- f)	f=true if n equals zero, else f=false	forth

Debug

.s	(–)	display the data stack depth and data stack on one line of the display	forth
dump	(addr n –)	display memory starting at addr, for n longs	forth
see	("text" --)	Attempt to decompile the word which follows in the input stream	forth
vlist	(--)	List the words in the context vocabulary (not chains) e.g. ‘Wifi vlist’ only lists words from the Wifi vocabulary	forth
words	(--)	List the words in the context vocabulary (including chains)	forth

See – example

Define a word:-

```
: demo    1 2 3 +  .  cr cr ;
```

'see demo' will show a decompiled view of the word to check for errors

```
: demo
   1 2 3 + . cr cr ;
```

Dictionary

>body	(xt -- addr)	addr is the data-field address corresponding to execution token xt	forth
>flags	(xt -- flags)	returns the flags of the word corresponding to execution token xt:- 1 if an immediate word 0 if a normal high level word 8 if a normal assembly language word	forth
>flags&			
>link	(xt -- addr)	addr is the link-field address corresponding to execution token xt	forth
>link&	(xt -- addr)	used internally by >link	forth
>name	(xt -- addr n | 0)	convert execution token xt to a name located at addr with n characters, else return 0 if not possible	forth
>params			
>size			
FIND	(addr n -- xt | 0)	using the counted string at addr with n characters, look a word up in the current dictionary stack, returning the execution token xt if found, else 0	forth
forget	("name" --)	find the word that follows in the input stream; if it exists in the current dictionary, remove it and all words that followed it from the dictionary and the corresponding compiled code	forth
here	(-- addr)	returns the address of the first free location above the code dictionary, where new words are compiled	forth
transfer	("name" --)	Move a word from its current dictionary to the current vocabulary. Useful for "hiding" helper words that aren't useful in normal programming	forth
transfer{	(--)	Move all the words that follow in the input stream up until a terminating } to the current vocabulary. All the words must have been defined beforehand	forth

Errors

abort	(–)	Raises an exception and interrupts the execution of the word and returns control to the interpreter	forth
abort”	(“ccc” –)	Stops general execution of the program. Displays an error message	forth

Exceptions

assert	(f --)	if flag f=true, execute throw	forth
catch	(xt -- err# | 0)	CATCH is very similar to EXECUTE except that it saves the stack pointers before EXECUTEing the guarded word at xt, removes the saved pointers afterwards, and returns a flag indicating whether or not the guarded word completed normally. 0=normal	forth
handler	(-- addr)	holds the return stack pointer for error handling - zero if no error occurred.	forth
throw	(err# --)	For any non-zero err#, throws the system back to CATCH so that the error condition can be processed. CATCH is backtracked by restoring the return stack from the pointer stored in 'handler' and popping the old handler and SP off the error frame on the return stack. So - 0 THROW does nothing	forth

See Appendix 3 for an explanation of catch and throw.

Files

From experiment - all filenames should be in lowercase.

BIN	(fam1 -- fam2)	Modify the implementation-defined file access method fam1 to additionally select a "binary", i.e., not line oriented, file access method, giving access method fam2.	forth
CLOSE-FILE	(fh -- ior)	Close the file identified by fileid. ior is the implementation-defined I/O result code.	forth
CREATE-FILE	(c-addr u fam -- fileid ior)	Create the file named in the character string specified by c-addr and u, and open it with file access method fam. The meaning of values of fam is implementation defined. If a file with the same name already exists, recreate it as an empty file.	forth
DELETE-FILE	(c-addr u -- ior)	Delete the file named in the character string specified by c-addr u. ior is the implementation-defined I/O result code.	forth
dump-file	(a1 n1 a2 n2 --)	a1,n1 text string to be saved in file: a2,n2 text string containing filename. Dump-file saves text to the spiffs using filename	forth
FILE-POSITION	(fileid -- ud ior)	ud is the current file position for the file identified by fileid. ior is the implementation-defined I/O result code. ud is undefined if ior is non-zero.	forth
FILE-SIZE	(fileid -- ud ior)	ud is the size, in characters, of the file identified by fileid. ior is the implementation-defined I/O result code. This operation does not affect the value returned by FILE- POSITION. ud is undefined if ior is non-zero.	forth
FLUSH-FILE	(fileid -- ior)	Attempt to force any buffered information written to the file referred to by fileid to be written to mass storage, and the size information for the file to be recorded in the storage directory if changed. If the operation is successful, ior is zero. Otherwise, it is an implementation-defined I/O result code.	forth
include	("name" --)	Using the next word in the input stream, name ...	forth
included	(c-addr u --)	Remove c-addr u from the stack. Save the current input source specification, including the current value of SOURCE-ID. Open the file specified by c-addr u, store the resulting fileid in SOURCE-ID, and make it the input source. Store zero in BLK. Other stack effects are due to the words included. Typical use: ... S" filename" INCLUDED	forth

needs	(“name” –)		forth
OPEN-FILE	(a n fam -- fh ior)	Open the file named in the character string specified by c-addr u, with file access method indicated by fam. The meaning of values of fam is implementation defined. If the file is successfully opened, ior is zero, fileid is its identifier, and the file has been positioned to the start of the file. Otherwise, ior is the implementation-defined I/O result code and fileid is undefined. Typical use: : X ... S" TEST.FTH" R/W OPEN-FILE ABORT" OPEN-FILE FAILED" ... ;	forth
R/O	(-- fam)	read only mode - used with OPEN-FILE, CREATE-FILE	forth
R/W	(-- fam)	read write mode - used with OPEN-FILE, CREATE-FILE	forth
READ-FILE	(c-addr u1 fileid -- u2 ior)	Read u1 consecutive characters to c-addr from the current position of the file identified by fileid. If u1 characters are read without an exception, ior is zero and u2 is equal to u1. If the end of the file is reached before u1 characters are read, ior is zero and u2 is the number of characters actually read. If the operation is initiated when the value returned by FILE-POSITION is equal to the value returned by FILE-SIZE for the file identified by fileid, ior is zero and u2 is zero. If an exception occurs, ior is the implementation-defined I/O result code, and u2 is the number of characters transferred to c-addr without an exception.	forth
remember	(--)	Save a snapshot to the default file (./myforth or /spiffs/myforth on ESP32)	forth
RENAME-FILE			forth
REPOSITION-FILE	(n fh -- ior) (ud fileid -- ior)	Reposition the file identified by fileid to ud. ior is the implementation-defined I/O result code. An ambiguous condition exists if the file is positioned outside the file boundaries. At the conclusion of the operation, FILE-POSITION returns the value ud.	forth
required	(a n –)		forth
RESET	(--)	Delete the default filename.	forth
RESIZE-FILE	(ud fileid -- ior)	Set the size of the file identified by fileid to ud. ior is the implementation-defined I/O result code.	forth

		If the resultant file is larger than the file before the operation, the portion of the file added as a result of the operation might not have been written. At the conclusion of the operation, FILE-SIZE returns the value ud and FILE- POSITION returns an unspecified value.	
restore	("name" --)	Restore a snapshot from a file	forth
revive	(--)	Restore the default filename	forth
save	("name" --)	Saves a snapshot of the current dictionary to a file	forth
startup:	("name" --)	Save a snapshot to the default file arranging for "name" to be run on startup	forth
W/O	(-- fam)	write only mode - used with OPEN-FILE, CREATE-FILE	forth
WRITE-FILE	(c-addr u fileid -- ior)	Write u characters from c-addr to the file identified by fileid starting at its current position. ior is the implementation-defined I/O result code. At the conclusion of the operation, FILE-POSITION returns the next file position after the last character written to the file, and FILE-SIZE returns a value greater than or equal to the value returned by FILE-POSITION.	forth

Floating Point Maths

Single precision floating-point support is available as a work in progress.
While initially left out in the name of minimalism, hardware support for floating-point argues some advantages to limited support.
Floating point numbers, denoted r below, are kept on a separate floating point stack.
NOTE: Tasks currently don't support floating point. A single floating point stack is shared by all tasks.

1/F	(r1 -- r2)	r2 = 1/r1	forth
AFLITERAL	(r --)	Compile r inline	forth
DOFLIT	(--)	Puts a float from the next cell onto float stack	forth
F-	(r1 r2 -- r3)	r3 = r1 - r2	forth
F.	(n--)	Display, with a trailing space, the top number on the floating-point stack using fixed-point notation:	forth
F.S	(--)	Print float stack	forth
F*	(r1 r2 -- r3)	r3 = r1 * r2	forth
F**	(r1 r2 -- r3)	r3 = r1 to the power r2	
F/	(r1 r2 -- r3)	r3 = r1 / r2	forth
F+	(r1 r2 -- r3)	r3 = r1 + r2	forth
F<	(r1 r2 -- flag)	flag = true if r1 < r2	forth
F<=	(r1 r2 -- r3)	flag = true if r1 <= r2	forth
F<>	(r1 r2 -- r3)	flag = true if r1 <> r2	forth
F=	(r1 r2 -- r3)	flag = true if r1 == r2	forth
F>	(r1 r2 -- r3)	flag = true if r1 > r2	forth
F>=	(r1 r2 -- r3)	flag = true if r1 >= r2	forth
F>S	(r -- n)	n = integer part of r (no rounding)	forth
F0<	(r -- f)	flag=true if r less than 0	forth
F0=	(r -- f)	flag-true if r == 0	forth
FABS	(r1 – r2)	r2 = absolute value of r1	forth
FATAN2	(r1 – r2)	r2 = atan (r1)	forth
FCONSTANT	(r "name")	creates a floating point constant	forth
FCOS	(r1 – r2)	r2 = cos (r1)	forth
FDROP	(r --)	drop r1 from the floating point stack	forth

FDUP	(r -- r r)	duplicate the top of floating point stack	forth
FEXP	(r1 – r2)	r2 = exp (r1)	forth
FLITERAL	(r --)	Compile r inline	forth
FLN	(r1 – r2)	r2 = ln (r1)	forth
FLOOR	(r1 – r2)	r2 = floor (r1)	forth
FMAX	(r1 r2 – r3)	R3 = the larger of r1 or r2	forth
FMIN	(r1 r2 – r3)	R3 = the smaller of r1 or r2	forth
FNEGATE	(r1 – -r1)	Negate r1	forth
FNIP	(ra rb -- rb)	Remove the 2nd item on the floating stack	forth
FOVER	(ra rb -- ra rb ra)	Copy the 2nd item on the floating stack to the top	forth
FP!	(a --)		forth
FP@	(-- a)		forth
FROT			forth
FSIN	(r1 – r2)	r2 = sin (r1)	forth
FSINCOS	(r1 – r2 r3)	r2 = sin (r1), r3 = cos (r1)	forth
FSQRT	(r r -- r)		forth
FSWAP	(ra rb -- rb ra)		forth
FVARIABLE	("name")	*create a floting point variable*	forth
PI	(-- r)		forth
precision	(– 6)	value – default = 6	forth
set-precision	(n –)	set the value of precision to n	forth
S>F	(n -- r)		forth
SF,	(r --)		forth
SF!	(r a --)	single precision store	forth
SF@	(a -- r)	single precision load	forth
SFLOAT	(-- 4)		forth
SFLOAT+	(a -- a+4)		forth
SFLOATS	(n -- n*4)		forth

HTTPD

notfound-response	(--)		httpd
bad-response	(--)		httpd
ok-response	(mime$ --)		httpd
response	(mime$ result$ status --)		httpd
send	(a n --)		httpd
path	(-- a n)		httpd
method	(-- a n)		httpd
hasHeader	(a n -- f)		httpd
handleClient			httpd
read-headers			httpd
completed?	(-- f)		httpd
body	(-- a n)		httpd
content-length	(-- n)		httpd
header	(a n -- a n)		httpd
crnl=	(n -- f)		httpd
eat	(n ch -- n a n)		httpd
skipover	(n ch -- n)		httpd
skipto	(n ch -- n)		httpd
in@<>	(n ch -- f)		httpd
end<	(n -- f)		httpd
goal#	(– a)	variable	httpd
goal	(– a)	variable	httpd
strcase=	(a n a n -- f)		httpd
upper	(ch -- ch)		httpd
server	(port --)		httpd
client-cr	(–)		httpd
client-emit	(ch --)		httpd
client-read	(-- n)		httpd

client-type	(a n --)		httpd
client-len	(– a)	variable	httpd
client	(– a)	sockaddr	httpd
httpd-port	(– a)	sockaddr	httpd
clientfd	(– n)	value, default = -1	httpd
sockfd	(– n)	value, default = -1	httpd
body-read	(– n)	value, default = 0	httpd
body-1st-read	(– n)	value, default = 0	httpd
body-chunk	(– a)	block of allotted data, body-chunk-size bytes long	httpd
body-chunk-size	(– 256)	constant	httpd
chunk-filled	(– n)	value, default = 0	httpd
chunk	(– a)	block of allotted data, chunk-size bytes long	httpd
chunk-size	(– 2048)	constant	httpd
max-connections	(– 1)	constant	httpd

Input / Output

adc	(pin# -- n)	alias for analogRead	forth
analogRead	(pin -- n)	Analog read from 0-4095	forth
dacWrite	(pin 0-255 --)	Write to DAC (pin 25, 26)	forth
digitalRead	(pin -- value)	Read GPIO state	forth
digitalWrite	(pin value --)	Set GPIO pin state	forth
pin	(value pin# --)	Set GPIO pin value e.g. HIGH 3 pin LOW 3 pin	forth
pinMode	(pin mode --)	Set GPIO pin mode e.g. 14 input pinMode \ set pin 14 as input	forth
pulseIn	(pin value usec -- usec/0)	Wait for a pulse	forth
tone	(channel freq)	Write tone frequency	Forth

PIN example (--) Demonstrates toggling GPIO23 at max rate using the PIN word

```
: maxtoggle
    23 output pinMode
    begin
      HIGH 23 pin
      LOW 23 pin
    key? Until ;
```

A scope showed that the pin remained high for 625nS or so. It was low for around 2.6uS because of the key? function slowing things up. So a reasonable speed, all considered

Read digital input example (gpiono --) Continuously display the state of a digital input until a key is pressed

```
: test ( gpiono -- )
    DUP INPUT pinMode          \ set 'gpiono' as an input
    begin
      DUP digitalRead . cr     \ display the state of 'gpiono'
    key? until                 \ until a key is pressed
    DROP ;                     \ stack tidy
```

Analogue input example (--) Demonstrates reading the voltage present on GPIO14 for 128 samples at 10 samples / s

```
100 value del                  \ this value determines the data sample rate
: delay del ms ;               \ time delay used to pace reading samples
: read          ( -- )
128 0 do                       \ Read 128 the input 128 times
    14 ADC . CR                \ Read the voltage on GPIO14
    DELAY                      \ And pause for 100 mS between each sample
loop
;
```

Interrupts

timer_isr_register	(group timer xt arg ret -- 0/err)		forth
esp_intr_alloc	(source flags xt args handle* -- 0/err)		interrupts
ESP_INTR_FLAG_DEFAULT	(-- 0)	Default handler allows per pin routing	interrupts
ESP_INTR_FLAG_EDGE	(-- 512)	gpio_install_isr_service flag	interrupts
ESP_INTR_FLAG_INTRDISABLED	(-- 2048)	gpio_install_isr_service flag	interrupts
ESP_INTR_FLAG_IRAM	(-- 1024)	gpio_install_isr_service flag	interrupts
ESP_INTR_FLAG_LEVELn	(n1 -- n2)	n2 = 2 to the power n1, gpio_install_isr_service	interrupts
ESP_INTR_FLAG_NMI	(-- 128)	gpio_install_isr_service flag	interrupts
ESP_INTR_FLAG_SHARED	(-- 256)	gpio_install_isr_service flag	interrupts
esp_intr_free	(handle -- 0/err)		interrupts
gpio_config	(gpio_config_t* -- 0/err)	GPIO common configuration. Configure GPIO's Mode,pull-up,PullDown,IntrType from a GPIO configure structure gpio_config_t	interrupts
gpio_deep_sleep_hold_dis	(--)	Disable all digital gpio pad hold function during Deep-sleep	interrupts
gpio_deep_sleep_hold_en	(--)	Enable all digital gpio pad hold function during Deep-sleep. When the chip is in Deep-sleep mode, all digital gpio will hold the state before sleep, and when the chip is woken up, the status of digital gpio will not be held. Note that the pad hold feature only works when the chip is in Deep-sleep mode, when not in sleep mode, the digital gpio state can be changed even you have called this function. Power down or call gpio_hold_dis will disable this function, otherwise, the digital gpio hold feature works as long as the chip enter Deep-sleep	interrupts
gpio_get_drive_capa	(pin cap* --		interrupts

bility	0/err)		
gpio_get_level	(pin -- level)	GPIO get input level of 'pin'	interrupts
gpio_hold_dis	(pin -- 0/err)		interrupts
gpio_hold_en	(pin -- 0/err)		interrupts
gpio_install_isr_service	(a --)	a = combination of gpio_install_isr_service flags - Install the driver's GPIO ISR handler service, which allows per-pin GPIO interrupt handlers Typically ESP_INTR_FLAG_DEFAULT	interrupts
gpio_intr_disable	(pin -- 0/err)	Disable GPIO module interrupt signal for 'pin'	interrupts
gpio_intr_enable	(pin -- 0/err)	Enable GPIO module interrupt signal for'pin'	interrupts
#GPIO_INTR_ANYEDGE	(-- 3)	constant - set interrupt for either +ve or -ve edge e.g. 2 #GPIO_INTR_ANYEDGE gpio_set_intr_type	interrupts
#GPIO_INTR_DISABLE	(-- 0)	constant - disable interrupt e.g. 2 #GPIO_INTR_DISABLE gpio_set_intr_type	interrupts
#GPIO_INTR_HIGH_LEVEL	(-- 5)	constant - e.g. 2 #GPIO_INTR_HIGH_LEVEL gpio_set_intr_type	interrupts
#GPIO_INTR_LOW_LEVEL	(-- 4)	constant - e.g. 2 #GPIO_INTR_LOW_LEVEL gpio_set_intr_type	interrupts
#GPIO_INTR_NEGEDGE	(-- 2)	constant - set interrupt on -ve edge e.g. 2 #GPIO_INTR_NEGEDGE gpio_set_intr_type	interrupts
#GPIO_INTR_POSEDGE	(-- 1)	constant - set interrupt on +ve edge e.g. 2 #GPIO_INTR_POSEDGE gpio_set_intr_type	interrupts
gpio_isr_handler_add	pin xt arg -- 0/err)	Having already set up the interrupt type, attach a new entry to the interrupt list, so that when the entry fires, the execution token 'xt' is called. If adding this entry was successful return true, else return an error code e.g. 2 ' myinterrupthandlerword 0 gpio_isr_handler_add	interrupts
gpio_isr_handler_remove	(pin -- 0/err)	Remove ISR handler for the corresponding GPIO pin	interrupts
gpio_pulldown_dis	(pin -- 0/err)	Disbale pull down load on 'pin'	interrupts
gpio_pulldown_en	(pin -- 0/err)	Enable pull down load on 'pin'	interrupts
gpio_pullup_dis	(pin -- 0/err)	Disable pull up load on 'pin'	interrupts
gpio_pullup_en	(pin -- 0/err)	Enable pull up load on 'pin'	interrupts
gpio_reset_pin	(pin -- 0/err)	Reset a gpio to default state (select gpio	interrupts

		function, enable pullup and disable input and output)	
gpio_set_direction	(pin mode -- 0/err)	Set gpio signal direction of 'pin'	interrupts
gpio_set_drive_capability	(pin cap -- 0/err)	Set GPIO pad 'pin' drive capability or strength 'cap'	interrupts
gpio_set_intr_type	(pin type -- 0/err)	Set the required i/o pin to the interrupt type, returning true if successful, else an error code	interrupts
gpio_set_level	(pin level -- 0/err)	GPIO set the output level pf 'pin' =1 or 0	interrupts
gpio_set_pull_mode	(pin mode -- 0/err)	Configure GPIO 'pin' pull-up/pull-down resistors by means of 'mode'. GPIO 34-39 don't have this facility	interrupts
gpio_uninstall_isr_service	(--)	Uninstall the driver’s GPIO ISR service, freeing related resources	interrupts
gpio_wakeup_disable	(pin -- 0/err)	Disable GPIO wake-up function on 'pin'	interrupts
gpio_wakeup_enable	(pin type -- 0/err)	Enable GPIO wake-up function on 'pin' - only type #GPIO_INTR_LOW_LEVEL or #GPIO_INTR_HIGH_LEVEL can be used	interrupts
pinchange	(xt pin --)	Call xt when pin changes e.g. 17 input pinMode : test ." pinvalue: " 17 digitalRead . cr ; ' test 17 pinchange	interrupts

Sense a digital input change using interrupt example (--) When the **'boot' button** is pressed or released, that change of state is reported on the display

```
interrupts

0 input pinmode                                  \ set GPIO0 as an input

: input. ( -- )                                  \ display the state of GPIO0
 ." GPIO0 input ="
 0 digitalRead . cr
;

: pinanyedge ( xt pin -- )                       \ add an ANYEDGE interrupt handler
 dup #GPIO_INTR_ANYEDGE gpio_set_intr_type throw
 swap 0 gpio_isr_handler_add throw
;

' input. 0 pinanyedge ( -- )                     \ If GPIO0 changes state, report it

\ This is just a demo - generally you want an interrupt to be as brief as possible
\ so printing from an interrupt routine is not recommended for real programs

\ N.B.The above example eventually will cause a system crash – presumably because of switch bounce causing an interrupt of the interrupt etc.
```

LED control – pulse width modulation

duty	(channel duty --)	like ledcWrite, with bounds check and scaling i.e. : duty 255 min 8191 255 */ ledcWrite ;	forth
freq	(channel freq --)	like ledcSetup, with scaling i.e. : freq (n n --) 1000 * 13 ledcSetup drop ;	forth
ledcAttachPin	(pin channel --)	Assigns which 'channel' (0-15) of the PWM engine the 'pin' is connected to	ledc
ledcDetachPin	(pin --)	Detaches #pin' from the pwm engine	ledc
ledcRead	(channel -- n)	Read the current dutycycle setting for 'channel'	ledc
ledcReadFreq	(channel -- freq)	Get frequency (x 1,000,000)	ledc
ledcSetup	(channel freq resolution -- freq)	Setup one of the 16 pwm channels (0-15) at frequency=freq*1000 Hz with 'resolution'	ledc
ledcWrite	(channel duty --)	Set 'channel' (0-15) at 'duty' level (0-100)	ledc
ledcWriteNote	(channel note octave	channel 0-12, octave 0-8, note 0-12 (note is as per the western musical scale)	ledc

	-- freq)		
ledcWriteTone	(channel freq -- n)	Write tone frequency (x 1000) e.g. 23 0 ledcAttachPin \ attach GPIO23 to channel 0 0 1000000 ledc WriteTone drop \ o/p 1kHz tine	ledc

Logic

AND	(n1 n2 – n3)	n3 = n1 AND n2 (bit wise)	forth
ARSHIFT	(n1 u – n2)	Arithmetic right shift of u bit-places on n1, giving n2	forth
invert	(n1 – n1')	Logical invert of all bits e.g. -1 is converted to 0, 2 is converted to -3	forth
LSHIFT	(x1 u -- x2)	Perform a logical left shift of u bit-places on x1, giving x2. Put zeroes into the least significant bits vacated by the shift. An ambiguous condition exists if u is greater than or equal to the number of bits in a cell.	forth
OR	(n1 n2 – n3)	n3 = n1 OR n2 (bit-wise)	forth
RSHIFT	(x1 u -- x2)	Perform a logical right shift of u bit-places on x1, giving x2. Put zeroes into the most significant bits vacated by the shift. An ambiguous condition exists if u is greater than or equal to the number of bits in a cell.	forth
XOR	(n1 n2 -- n3)	n3 = n1 XOR n2 (bit-wise)	forth

Looping

?do	(n1 n2 --)	e.g. : test ?do i . loop ; if n1 = n2, then inside ?do .. loop is NOT executed if n1 <> n2, it behaves like do .. loop	forth
+loop	(n --)	e.g. : +looptest 20 0 do i . 2 +loop ; results when run in 0 2 4 6 8 10 12 14 16 18 being displayed	forth
again	(--)	begin <forth words> again	forth
begin	(--)	begin <forth words> until	forth
do	(n1 n2 --)	: test 10 4 do i . loop ; produces 4 5 6 7 8 9 on the display	forth
EXIT	(--)	Return control to the calling definition. Before executing EXIT within a do-loop, a program shall discard the loop-control parameters by executing 'unloop'	forth
for	(n --)	: test 10 for i . next ; produces 10 9 8 7 6 5 4 3 2 1 0 on display n.b. n will make the for loop run n+1 times by design	forth
i	(-- n)	Place current loop index on top of stack	forth
j	(-- n)	Place index count for next outer lop top of stack	forth
leave	(--)	Force do loop termination	forth
loop	(--)	part of do <forth words> loop construct	forth
next	(--)	part of for <forth words> next construct	forth
repeat	(--)	part of begin <forth words that leave f on the stack> while < more forth words> repeat	forth
unloop	(--)	Discard the loop-control parameters for the current nesting level e.g. Typical use: : unlooptest <limit> <first> DO ... test IF ... UNLOOP EXIT THEN ... LOOP ... ;	forth
until	(f --)	begin <forth words that leave f on stack> until	forth
while	(f --)	part of begin <forth words that leave f on the stack> while < more forth words> repeat	forth

unloop example Demonstrates a loop being terminated before completion by UNLOOP

```
: test
1000 0 do                          \ start  a 1000 long loop
key? if                            \ if a  key was pressed
       key drop                    \ read the key and drop it
       cr ." loop terminating "
       unloop                      \ get rid of the loop
       exit                        \ return to the calling word
      then
cr ." loopcount = " i .            \ show us the loop count
500 ms                             \ loop every 1/2 s
loop
;
```

Maths

-	(n1 n2 -- n3)	n3 = n1 - n2	forth
*****	(n1 n2 -- n3)	n3 = n1 * n2	forth
***/**	(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	forth
***/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	forth
/	(n1 n2 -- n3)	Divide n1 by n2, giving the single-cell quotient n3	forth
/mod	(n1 n2 -- n3 n4)	Divide n1 by n2, giving the single-cell remainder n3 and the single-cell quotient n4	forth
+	(n1 n2 -- n3)	n3 = n1 + n1	forth
2*	(n1 -- n2)	n2 = n1 * 2	forth
2/	(n1 -- n2)	n2 = n1 / 2	forth
4*	(n1 -- n2)	n2 = n1 * 4	forth
4/	(n1 -- n2)	n2 = n1 / 4	forth
abs	(n1 -- n2)	n2 is the absolute value of n1	forth
max	(n1 n2 -- n3)	n3 = the larger of n1 or n2	forth
min	(n1 n2 –	n3 = the smaller of n1 or n2	forth

	n3)		
mod	(n1 n2 – n3)	Divide n1 by n2, giving the single-cell remainder n3	forth
negate	(n – -n)	Two's complement of top of stack	forth
U/MOD	(u1 u2 -- rem quot)	Unsigned division, leaving remainder and quotient	forth
1+	(n-- n+1)	increment value on the stack by 1	forth
1-	(n-- n1-)	decrement value on the stack by 1	forth

Memory

ESP32forth stores numbers in little-endian format (the LS byte of the number is place at the lowest address in memory) unless noted otherwise (e.g. in ‘sockets’)

!	(n addr -->)	store x at addr	forth
@	(addr --n)	Retrieves the integer value n stored at address addr	forth
+!	(n addr --)	Increments the content of a variable by the value n	forth
+to	(n "valuename" --)	Adds n to the value whose name follows in the input stream e.g. 2 value myvalue \ define a value called myvalue = 2 3 +to myvalue \ myvalue now equals 5	forth
to	(n --)	Change a value e.g. 10 value myvalue \ set up myvalue = 10 5 to myvalue \ myvalue now returns 5	forth
2!	(n1 n2 addr -->)	store n1, n2 as a 64 bit value at addr	forth
2@	(addr --n1 n2)	read n1, n2 as a 64 bit value from addr	forth
C!	(c addr -->)	store byte c at addr	forth
C@	(addr --c)	read byte c from addr	forth
allocate	(n -- a ior)	reserve a memory chunk of n bytes, returns the start address, ior=true if successful, else false - see also free and resize	forth
blank	(addr n –)	Fill block starting at addr for n bytes with $20	
cell/	(n1 -- n2)	n2 = n1 / 4	forth
cell+	(n1 -- n2)	n2 = n1 + 4	forth
cells	(n – n*4)	Conversion, cells to bytes	forth

cmove	(addr1 addr2 n --)	move n bytes from addr1 to addr2, starting at addr1 and proceeding toward high memory	forth
cmove>	(addr1 addr2 n --)	copy n bytes from memory starting at addr1 to that starting at addr2, proceeding from higher addresses to lower addresses	forth
erase	(addr n –)	Fill block starting at addr for n bytes with $00	forth
fill	(addr n c)	fill memory from addr for n bytes with byte c	forth
free	(a --)	free memory previously reserved allocate - see also resize	forth
L!	(n addr --)	write n to addr in real ESP32 memory space	forth
resize	(a n -- a ior)	ior=true if a memory chunk reserved with MALLOC resized to n bytes correctly, else ior=false - see also allocate and free. The high-address end is adjusted. When increasing the size, all data is preserved. High end-data is not initialised. When decreasing the size, high-end address data may be lost	forth
SL@	(addr --n)	read signed long n from real ESP32 memory space	forth
SW@	(addr – sw)	Read signed word from real ESP32 memory space	forth
UL@	(addr --u)	read unsigned long u from real ESP32 memory space	forth
UW@	(addr – uw)	Read unsigned word uw from real ESP32 memory space	forth
W!	(w addr –)	Store word w at address addr	forth

Number I/O

?	(addr --)	display value at addr	forth
.	(n –)	display top of stack	forth
#	(u1 -- u2)	Convert next digit of u1 and HOLD it	forth
#>	(u -- addr n)	Drop u and prepare string for TYPE	forth
#s	(u -- 0)	Convert and HOLD all remaining significant digits	forth
<#	(--)	Begin a formatted number conversion	forth
base	(-- addr)	Stores the current number display base - defaults to 10 decimal	forth
binary	(–)	Set current number base to 2 decimal	forth
decimal	(–)	Set current number base to 10 decimal	forth
extract	(n base -- n c)	extracts the least significant digit from a number n. n is divided by the radix in BASE and returned on the stack	forth
hex	(–)	Set current number base to 16 decimal	forth
hold	(c --)	Insert character into formatted string	forth
n.	(n --)	display n in decimal, regardless of current number base	forth
octal	(--)	Set current number base to 8 decimal	forth
pad	(-- addr)	returns the address of the text buffer where numbers are constructed and text strings are stored temporarily	forth
sign	(n --)	HOLD minus sign only if n is negative	forth
u.	(u --)	Display u unsigned in the current number base	forth
f.	(r –)	Display the top number on the floating point stack	forth

Formatted Number Conversion example (n –) *Demonstrates printing numbers in special formats*

```
: dollars.
dup abs                 \ duplicate and take absolute value
<# # #                  \ start the formatted number conversion and convert the cents
46 hold                 \ insert a decimal point character
#s                      \ convert the dollars
36 hold                 \ add a $ character
swap sign               \ if the number is negatve, add a - character
#> type ;              \ complete the formatted number conversion

--> 123456 dollars.
$1234.56 ok
--> -123456 dollars.
-$1234.56 ok
```

OLED display

To include the following words requires a change to the source code:-

#define ENABLE_OLED_SUPPORT	#define ENABLE_OLED_SUPPORT to activate the lib.
# include <Adafruit_GFX.h>	install in your system the Adafruit libraries
# include <Adafruit_SSD1306.h>	install in your system the Adafruit libraries

OledInit	(--)	initialize the display to accept commands	oled
OledDelete	(n -- a)		oled
OledBegin	(n -- a)	initialization	oled
OledHOME	(n -- a)	send cursor to upper left home position	oled
OledCLS	(n -- a)	clears the display	oled
OledTextc	(n -- a)		oled
OledPrintln	(n -- a)	print a zero= null terminated string from the stack	oled
OledNumln	(n -- a)	print + CR a number on the stack (int)	oled
OledNum	(n --)	print a number on the stack (int)	oled
OledDisplay	(--)	show the buffer on the OLED display	oled
OO	(n -- a)	is an abbreviation of Oleddisplay	oled
OledPrint	(z" " addr --)	print a zero= null terminated string from the stack	oled
OledInvert	(--)	invert the background & foreground colors	oled
OledTextsize	(n --)		oled
OledSetCursor	(n --)		oled

OledPixel	(x y C --)	draw a pixel x y coordinates C= Color , f.e.use 1	oled
OledDrawL	(x y x2 y2 1 --)	draw a line x y x2 y2 coordinates C= Color , f.e. 1	oled
OledCirc	(x y r 1 --)	draw a circle	oled
OledCircF	(x y r 1 --)	draw a circle filled	oled
OledRect	(x y x2 y2 1 --)	draw a rectangle	oled
OledRectF	(x y x2 y2 1 --)	draw a rectangle filled	oled
OledRectR	(x y x2 y2 1 --)	draw a rectangle rounded edges	oled
OledRectRF	(n -- a)	draw a rectangle rounded edges filled	oled
OledAddr	(n -- a)	address of I2C device	oled
OledNew	(n -- a)		oled

OLED example (–) *Demonstrates how to talk to the OLED 128x64 pixels*

```
Oled
OLEDINIT                    \ initalise display / greeting message
123 Olednum Oleddisply      \ print the number top of stack
Oledhome oledcls OO          \ home the cursor, clear the screen
1 1 35 35 1 Oledrect OO      \ draw rectangle
Oledinvert OO                \ invert the display
```

Registers

m!			registers
m@			registers

RMT Remote Control Transceiver

rmt_set_clk_div	(channel div8 -- err)		RMT
rmt_get_clk_div	(channel @div8 -- err)		RMT
rmt_set_rx_idle_thresh	(channel thresh16 -- err)		RMT
rmt_get_rx_idle_thresh	(channel @thresh16 -- err)		RMT

rmt_set_mem_block_num	(channel memnum8 -- err)		RMT
rmt_get_mem_block_num	(channel @memnum8 -- err)		RMT
rmt_set_tx_carrier	(channel enable highlev lowlev carrierlev -- err)		RMT
rmt_set_mem_pd	(channel f -- err)		RMT
rmt_get_mem_pd	(channel @f -- err)		RMT
rmt_tx_start	(channel f -- err)		RMT
rmt_tx_stop	(channel -- err)		RMT
rmt_rx_start	(channel f -- err)		RMT
rmt_rx_stop	(channel -- err)		RMT
rmt_tx_memory_reset	(channel -- err)		RMT
rmt_rx_memory_reset	(channel -- err)		RMT
rmt_set_memory_owner	(channel owner -- err)		RMT
rmt_get_memory_owner	(channel @owner -- err)		RMT
rmt_set_tx_loop_mode	(channel f -- err)		RMT
rmt_get_tx_loop_mode	(channel @f -- err)		RMT
rmt_set_rx_filter	(channel enable thresh8 -- err)		RMT

rmt_set_source_clk	(channel baseclk -- err)		RMT
rmt_get_source_clk	(channel @baseclk -- err)		RMT
rmt_set_idle_level	(channel enable level -- err)		RMT
rmt_get_idle_level	(channel @enable @level -- err)		RMT
rmt_get_status	(channel @status -- err)		RMT
rmt_set_rx_intr_en	(channel enable -- err)		RMT
rmt_set_err_intr_en	(channel enable -- err)		RMT
rmt_set_tx_intr_en	(channel enable -- err)		RMT
rmt_set_tx_thr_intr_en	(channel enable thresh -- err)		RMT
rmt_set_gpio	(channel mode gpio# invertsig -- err)		RMT
rmt_config	(rmt_config_t*)		RMT
rmt_isr_register	(fn arg allocflags handle -- err)		RMT
rmt_isr_deregister	(handle -- err)		RMT
rmt_fill_tx_items	(channel @items items# offset		RMT

	-- err)		
rmt_driver_install	(channel rxbufsize allocflags -- err)		RMT
rmt_driver_uinstall	(channel -- err)		RMT
rmt_get_channel_status	(channel @status -- err)		RMT
rmt_get_counter_clock	(channel @clockhz -- err)		RMT
rmt_write_items	(channel @items items# wait -- err)		RMT
rmt_wait_tx_done	(channel time -- err)		RMT
rmt_get_ringbuf_handle	(channel @handle -- err)		RMT
rmt_translator_init	(channel fn -- err)		RMT
rmt_translator_set_context	(channel @context -- err)		RMT
rmt_translator_get_context	(channel @@context -- err)		RMT
rmt_write_sample	(channel src src# wait -- err)		RMT
rmt_register_tx_end_callback		NOT SUPPORTED	RMT
rmt_memory_rw_rst		DEPRECATED USE rmt_tx_memory_reset or rmt_rx_memory_reset	RMT
rmt_set_intr_enable_mask		DEPRECATED interrupt handled by driver	RMT
rmt_clr_intr_enable_mask		DEPRECATED interrupt handled by driver	RMT

rmt_set_pin		DEPRECATED use rmt_set_gpio instead	RMT

RTOS support

rtos-builtins			RTOS
vTaskDelete			RTOS
xPortGetCoreID			RTOS
xTaskCreatePinnedToCore			RTOS

SD Card

SD.begin	(--)	uses all the defaults "/sd" etc.	SD
SD.beginDefaults	(-- sspin SPIClass frequency mountpointsz maxfiles format_if_empty)	(SS SPI 4000000 "/sd" 5 false)	SD
SD.beginFull	(sspin SPIClass frequency mountpoint maxfiles format_if_empty --)		SD
SD.end	(--)		SD
SD.cardType	(-- n)		SD
SD.totalBytes	(-- n)		SD
SD.usedBytes	(-- n)		SD

SD_MMC Card

SD_MMC.begin	(mount mode1bit)	default mode1bit=false	SD_MMC
SD_MMC.beginDefaults			SD_MMC
SD_MMC.beginFull			SD_MMC
SD_MMC.cardType	(-- n)		SD_MMC
SD_MMC.end	(--)		SD_MMC
SD_MMC.totalBytes	(-- n)		SD_MMC
SD_MMC.usedBytes	(-- n)		SD_MMC

Serial communication

‘Serial’ is the default forth terminal, pin GPIO1 is data out from the ESP32, GPIO3 is data in.
‘Serial2’ is unused by default. Pin GPIO17 is data transmit from the ESP32, GPIO16 is data in.

Serial.available **Serial2.available**	(-- f)	Get the number of bytes (characters) available for reading from the serial port. This is data that’s already arrived and stored in the serial receive buffer (which holds 64 bytes)	serial
Serial.begin **Serial2.begin**	(baud --)	Start serial port. Sets the data rate in bits per second (baud) for serial data transmission	serial
Serial.end **Serial2.end**	(--)	Disables serial communication, allowing the RX and TX pins to be used for general input and output. To re-enable serial communication, call Serial.begin	serial
Serial.flush **Serial2.flush**	(--)	Waits for the transmission of outgoing serial data to complete	serial
Serial.readBytes **Serial2.readBytes**	(a length -- n)	Serial.readBytes reads characters from the serial port into a buffer, address a. The function terminates if the determined length has been read. The number of bytes, n is returned	serial
Serial.write **Serial2.write**	(a n -- n)	Writes n bytes of data to the serial port from buffer at address a	serial

Serial Bluetooth

esp_bt_dev_get_address	*(-- a)*	*addr of 6 byte mac address*	*bluetooth*

Serial Bluetooth:-

SerialBT.new	(-- bt)	Allocate new BT objectbluetooth	bluetooth
SerialBT.delete	(bt --)	Free BT object	bluetooth
SerialBT.begin	(localname ismaster bt -- f)		bluetooth
SerialBT.end	(bt --)		bluetooth
SerialBT.available	(bt -- f)		bluetooth
SerialBT.readBytes	(a n bt -- n)		bluetooth
SerialBT.write	(a n bt -- n)		bluetooth
SerialBT.flush	(bt --)		bluetooth
SerialBT.hasClient	(bt -- f)		bluetooth

SerialBT.enableSSP	(bt --)		bluetooth
SerialBT.setPin	(z bt -- f)		bluetooth
SerialBT.unpairDevice	(addr bt -- f)		bluetooth
SerialBT.connect	(remotename bt -- f)		bluetooth
SerialBT.connectAddr	(addr bt -- f)		bluetooth
SerialBT.disconnect	(bt -- f)		bluetooth
SerialBT.connected	(timeout bt -- f)		bluetooth
SerialBT.isReady	(checkMaster timeout -- f)		bluetooth

SPI FLASH memory

esp_partition_check_identity			spi_flash
esp_partition_erase_range			spi_flash
esp_partition_find			spi_flash
esp_partition_find_first			spi_flash
esp_partition_get			spi_flash
esp_partition_get_sha256			spi_flash
esp_partition_iterator_release			spi_flash
esp_partition_mmap			spi_flash
esp_partition_next			spi_flash
esp_partition_read			spi_flash
esp_partition_t			spi_flash
esp_partition_t_size			spi_flash

esp_partition_verify			spi_flash
esp_partition_write			spi_flash
list-partition-type			spi_flash
list-partitions			spi_flash
p.			spi_flash
p>address			spi_flash
p>gap			spi_flash
p>label			spi_flash
p>size			spi_flash
p>subtype			spi_flash
p>type			spi_flash
spi_flash_cache_enabled			spi_flash
spi_flash_cache2phys			spi_flash
spi_flash_erase_range			spi_flash
spi_flash_erase_sector			spi_flash
spi_flash_get_chip_size			spi_flash
spi_flash_init			spi_flash
spi_flash_mmap			spi_flash
spi_flash_mmap_dump			spi_flash
spi_flash_mmap_get_free_pages			spi_flash
spi_flash_munmap			spi_flash
spi_flash_phys2cache			spi_flash
spi_flash_read			spi_flash
spi_flash_read_encrypted			spi_flash

spi_flash_write			spi_flash
spi_flash_write_encrypted			spi_flash
spi_flash-builtins			spi_flash
SPI_PARTITION_SUBTYPE_ANY			spi_flash
SPI_PARTITION_TYPE_APP			spi_flash
SPI_PARTITION_TYPE_DATA			spi_flash

Serial Peripheral Interface Flash File System (SPIFFS)

SPIFFS.begin	(format-on-fail path-z max-files -- f)	Mounts file system. It must be called before any other SPIFFS words are used. Returns true if file system was mounted successfully, false otherwise. If format-on-fail is true the 'disk' will be formatted	SPI filesystem
SPIFFS.end	(--)	Unmounts the flash memory file system	SPI filesystem
SPIFFS.format	(-- f)	Format the flash memory 'disk'. returns true if successful	SPI filesystem
SPIFFS.totalBytes	(-- n)	Returns the total capacity of the flash memory 'disk'	SPI filesystem
SPIFFS.usedBytes	(-- n)	Returns the total space occupied by flash memory files	SPI filesystem

SPIFF File Utilities

The following commands can be used to manipulate files on the SPIFFS drive:-

cp	(“src” “dst” –)	Copy source to destination file e.g. cp /spiffs/test1.fs /spiffs/test2.fs	forth
mv	(“src” “newname” –)	Rename source file to newname file e.g mv /spiffs/test1.fs /spiffs/test2.fs	forth
rm	(“path” –)	Remove “path” file e.g. rm /spiffs/test1.fs	forth
touch	(“path” –)	Create “path” file if it doesn’t exist e.g. touch /spiffs/test1.fs	forth
cat	(“path” –)	Display contents of “path” file e.g. cat /spiffs/test1.fs	forth
ls	(“path” –)	List files or directories in “path” e.g. ls /spiffs	forth

Sockets

Addrlen is a parameter in quite a few Sockets words. Unless otherwise noted, addrlen is just the number 16 defining the length in bytes of a sockaddr structure for IPv4. When calling words **sockaccept** and **recvfrom**, addrlen must be the address of a variable, so that these two functions can update the value.

->addr!	(n a --)	set big-endian address in sockaddr	sockets
->addr@	(a -- n)	get big-endian address from sockaddr	sockets
->h_addr	(hostent – a)	Get host address from a hostent structure (returned by gethostbyname)	sockets
->port!	(n a --)	set port in sockaddr	sockets
->port@	(a -- n)	get port from sockaddr	sockets
ip.	(n --)	Print address as x.y.z.w IP address.	sockets
ip#		Part of ip.	sockets
AF_INET	(-- 2)	constant	sockets
bind	(sock addr addrlen -- 0/err)	The bind function assigns an address to an unnamed socket. Sockets created with socket() function are initially unnamed; they are identified only by their address family	sockets
bs,	(n –)	Compile 16 bit number (as 2 bytes) into a definition, big-endian (ms byte in lowest address)	sockets
connect	(sock addr addrlen -- 0/err)	The connect function requests a connection to be made on a socket. N.B. A socket is closed again with <sock> CLOSE-FILE	sockets
errno	(-- err)	f = the error number of the last sockets action	sockets
gethostbyname	(hostnamez -- hostent/0)		sockets
l,	(n –)	Compile 32 bit number n (as 4 bytes) into a definition	sockets
listen	(sock backlog -- 0/err)	Listen for socket connections and limit the queue of incoming connections - 'backlog' is the max no. of connections that can be put on hold	sockets
NON-BLOCK	(sock – 0/err)	Certain words in the socket, file or serial port vocabs normally block until complete. NON-BLOCK converts these to non-blocking. If a function would have normally blocked, it now returns -1 and errno is set to ESP_ERR_WIFI_WOULD_BLOCK = $300E	forth
poll	(pollfds n	A file descriptor for a socket that is listening for	sockets

	timeout -- fd/err)	connections will indicate that it is ready for reading, once connections are available. A file descriptor for a socket that is connecting asynchronously will indicate that it is ready for writing, once a connection has been established.	
recv	(sock a n1 flags -- n2/err)	receive data to buffer at address a, required size n1. returns number of bytes actually read n2. It's up to you to call recv again until all the data is read. Will block if the receive buffer is empty. Use NON-BLOCK to alter that	sockets
recvfrom	(sock a n1 flags addr addrlen -- n2/err)	Works like recv, albeit addr , addrlen is a sockaddr showing who sent the message. N.B. make sure here that addrlen is an address of a variable, not a number ! Will block if the receive buffer is empty. Use NON-BLOCK to alter that	sockets
recvmsg	(sock msg flags -- n/err)	Will block if the receive buffer is empty. Use NON-BLOCK to alter that	sockets
s,		Compile 16 bit number (as 2 bytes) into a definition	sockets
select	(numfds readfds writefds errfds timeout -- fd/err)	A file descriptor for a socket that is listening for connections will indicate that it is ready for reading, when connections are available. A file descriptor for a socket that is connecting asynchronously will indicate that it is ready for writing, when a connection has been established	sockets
send	(sock a n1 flags -- n/err)	send data at address a, n1 bytes long. Return n2 the actual number of bytes received or -1 as an error flag. Will block if the transmit buffer is full. Use NON-BLOCK to alter that	sockets
sendmsg	(sock msg flags -- n/err)	Returns the actual number of bytes received or -1 as an error flag. Will block if the transmit buffer is full. Use NON-BLOCK to alter that	sockets
sendto	(sock a n flags addr addrlen -- n/err)	Like send, but the recipient is specifically addressed with sockaddr addr, addrlen. Return the actual number of bytes received or -1 as an error flag	sockets
setsockopt	(sock level optname optval optlen -- 0/err)	The setsockopt() function sets the option specified by the option_name argument, at the protocol level specified by the level argument, to the value pointed to by the option_value argument for the socket	sockets

		associated with the file descriptor specified by the socket argument.	
sizeof(sockaddr_in)	(-- 16)	Constant – the size of a standard IPv4 sockaddr is 16 bytes	sockets
SO_REUSEADDR	(– 2)	constant	sockets
SOCK_DGRAM	(– 2)	constant	
SOCK_RAW	(– 3)	constant	
SOCK_STREAM	(-- 1)	constant	sockets
sockaccept	(sock1 addr addrlen -- sock2/err)	The sockaccept function extracts the first connection on the queue of pending connections, creates a new socket sock2 with the same socket type protocol and address family as the specified socket, and returns a new file descriptor for sock2 in sockaddr addr, addrlen. N.B. make sure here that addrlen is an address of variable, not a number! It must be set to 16 (sockaddr size in bytes) before calling sockaccept. Normally blocks until a remote client connects. Use NON-BLOCK to alter that	sockets
sockaddr	("name" --)	creates a sockaddr structure	sockets
socket	(domain type protocol – sock/err)	domain is usually AF_INET for tcpip using a 4 byte internet address. type is SOCK_STREAM for tcp or SOCK_DGRAM for udp; protocol is 0 for internet: returns a reference sock	sockets
sockets-builtins			sockets
SOL_SOCKET	(– 1)	constant	sockets

Stack functions

-rot	(a b c – c a b)	rotate top cell to 3rd	forth
?DUP	(x -- 0 | x x)	Duplicate x if it is non-zero.	
>R	(n –)	move n to the return stack	forth
2drop	(n1 n2 –)	discard the top of stack	forth
2dup	(n1 n2 – n1 n2 n1 n2)	duplicate the top two items on the data stack	forth
depth	(– n)	return the data stack depth on the top of stack e.g. 3 2 1 depth displays 3 2 1 3	forth

DROP	(n –)	discard the top of stack	forth
DUP	(n – n n)	duplicate the top of stack	forth
f.s	(–)	displays the floating point number stack	forth
nip	(n1 n2 -- n2)	remove the 2nd item on the data stack	forth
OVER	(n1 n2 -- n1 n2 n1)	duplicate 2nd item on the data stack	forth
R@	(-- n)	copy the top of the return stack to the top of data stack	forth
R>	(– n)	Move top of return stack to data stack	forth
rdrop	(--)	drop the top of the return stack	forth
rot	(a b c -- b c a)	rotate 3rd cell to top	forth
RP!	(addr --)	set the return stack pointer	forth
RP@	(-- addr)	read the return stack pointer	forth
rp0	(-- addr)	constant - the initial value of the return stack pointer at switch-on	forth
SP!	(addr --)	set the data stack pointer	forth
SP@	(-- addr)	read the data stack pointer	forth
sp0	(-- n)	constant - the initial value of the data stack pointer at switch-on	forth
SWAP	(n1 n2 – n2 n1)	swap the top two data stack entries	forth

Streams

These words enable the creation of first-in first-out buffers or queues of chrs or bytes, useful where a stream of data is to be handled. The words are multitask compatible.

>offset	(n st -- a)	internal word	streams
>read	(st -- rd)	internal word	streams
>stream	(a n st --)	read string, a n , from stream. Terminate when n chrs received or stream empty	streams
>write	(st -- wr)	internal word	streams
ch>stream	(ch st --)	wait until there is space, then write one char	streams
empty?	(st -- f)	returns true if stream empty	streams
full?	(st -- f)	returns true if stream full	streams
stream	(n "name" --)	define a stream, size n, using the next word in the input stream as the name e.g. 200 stream	streams

		myinputstream 20000 stream myoutputstream	
stream#	(st -- n)	returns the number of chrs waiting to be read in stream st	streams
stream>	(a n st --)	send string, a n , to stream st	streams
stream>ch	(st -- ch)	wait until data is available then read one char	streams
wait-read	(st --)	wait until there is data ready to read from st	streams
wait-write	(st --)	wait until there is space to write to stream st	streams

String functions

ESP32forth supports both null terminated strings used in calling the various C based vocabularies and counted strings as used by many forth systems.

[char]	(--)	compile the first letter of the following word in the definition e.g. : my-char [char] ALPHABET emit char emit ; executing my-char fred will display:- Af	forth
r"	("string" -- a n)	Creates a temporary counted string	forth
r|	(string| -- a n)	Creates a temporary counted string ending with |	forth
s"	(-- addr cnt)	Creates a zero terminated string. Leaves the string address addr and the character count cnt on the stack e.g. s" Hello Bob"	forth
s>z	(a n -- z)	Convert a counted string string to null terminated string	forth
startswith?	(addr1 n1 addr2 n2 -- f)	f=true if string at addr1 starts with string at addr2, else f=false	forth
str	(n -- addr cnt)	convert n to a counted string	forth
str=	(addr1 n1 addr2 n2 -- f)	f=true if the two counted strings are equal, else f=false	forth
z"	("string" -- addr)	Creates a null terminated string on the heap at addr	forth
z>s	(addr -- addr n)	Convert a null terminated string at addr to a counted string	forth

r| example (--) Demonstrates making a standalone application with the aid of of a temporary string made with r|

```
r| z" NETWORK-NAME" z" PASSWORD" webui |      \ create a string to start the web server
s" /spiffs/autoexec.fs" dump-file             \ save it to file autoexec.fs
```

Structures

align-by	(a1 n -- a2)	Adjust address a up to an n byte boundary	structures
field	(n "name")	Define a field in the structure	structures
i16	(– 2)	Used to define a 16 bit field	structures
i32	(– 4)	Used to define a 32 bit field	structures
i64	(– 8)	Used to define a 64 bit field	structures
i8	(– 1)	Used to define a 8 bit field	structures
last-align	(– a)	variable	structures
last-struct	(– a)	variable	structures
long	(– 4)	Used to define a 32 bit field for a long	structures
ptr	(– 4)	Used to define a pointer field of one cell	structures
struct	("name") (– total)	Start a structure definition. When “name” is execute it returns the total byte count of the structure	structures
struct-align	(n --)	Set the last structure defined to be on an n byte boundary	structures
typer	(align sz "name")	define a field type with alignment ‘align’ and ‘sz’ bytes in size	structures

STRUCTURES example (--) Demonstrates definition and use of a structure

First a recipe for a structure is defined:-

```
struct timer
i32 field counter
i32 field limit
```

Now we create a variable based on that structure:-

```
create mytimer timer allot
```

Now we can access fields within that variable:-

```
mytimer limit @       20 mytimer counter !
```

System

bye	(--)	deferred word, defaults to esp32-bye	forth
CELL	(-- 4)	returns the number of bytes per standard forth number - 32 bits	forth
echo	(-- addr)	All input stream is echoed on output stream if echo = -1, else only partial echo (--> ok and errors) if set 0	forth
evaluate	(addr cnt --)	evaluate the counted string at addr, as if typed in at the command line	forth
EXECUTE	(xt --)	Execute the word whose execution token is top of stack	forth
HIGH	(-- 1)	Logic high is represented by 1 e.g. HIGH 1 pin	forth
hld	(-- adr)	holds a pointer in building a numeric output string	forth
INPUT	(-- 1)	constant used to set a pin as an input e.g. 2 INPUT pinMode	forth
LED	(-- 2)	Some ESP32 modules have an LED fitted on GPIO pin 2	forth
LOW	(-- 0)	Logic low is represented by 0 e.g. LOW 4 pin	forth
OUTPUT	(-- 2)	constant used to set a pin as an output e.g. 4 OUTPUT pinMode	forth
quit	(--)	Leave stack intact, but return control to input stream	forth
state	(-- addr)	system variable, state=true system is interpreting, state=false system is compiling	forth
TERMINATE	(n --)	Call system exit	forth

Tasks – multitasking

.tasks	(--)	List running tasks	tasks
pause	(--)	yield to other tasks	forth
start-task	(task --)	Activate a task	forth
task	(xt dsz rsz "name" --)	Create a new task, named using the next word in the input stream, with 'dsz' size data stack, and 'rsz' size return stack, execution to start at 'xt' execution token	forth

main-task			tasks
task-list	(– addr)	Return the start address of the task list	tasks

Tasks example (--) Demonstrates adding a 10 second timer task

```
: hi begin ." Time is: " ms-ticks . cr 10000 ms again ;          \ Print the tick every 10sec.
' hi 100 100 task my-counter                                     \ define my-counter task
my-counter start-task                                            \ start the my-counter task

\ in between print outs, type tasks .tasks <cr> and observe the active tasks are
\ main-task my-counter yield-task
```

Actually the word ‘hi’ above could have been written as:-

```
: hi ." Time is: " ms-ticks . cr 10000 ms ;                      \ Print the tick and pause 10s
```

The word ‘task’ compiles:-

```
again:  xt of ‘hi’
        pause
        branch to again
```

So any word written as a ‘one-shot’ will in fact repeat when assigned to a task and a ‘pause’ is already built in to ensure task switching

Telnet

broker	(--)	Deferred word - executes broker-connection by default	telnetd
broker-connection	(--)	Processing loop for the active TELNET link	telnetd
client			telnetd
client-len	(-- adr)	variable	telnetd
clientfd	(-- flag)	value, default to -1	telnetd
connection			telnetd
server	(port --)	Start telnet server daemon on port	telnetd
sockfd	(-- flag)	value, default to -1	telnetd
telnet-emit	(c --)	Emit c character on the active telnet port	telnetd
telnet-emit'			telnetd
telnet-key	(-- c)	Retrieve a character c from the active telnet port	telnetd
telnet-port	(a n --)		telnetd
telnet-type	(adr len --)	Send a counted string on the active telnet port	telnetd

Telnet example (--) Demonstrates starting the telnet server to enable terminal communication with ESP32forth over WiFi

```
z" yourrouterid" z" yourpassword" login cr \ Login to your Wifi router
telnetd                                   \ vocabulary TELNET
552 server                                \ start the telnet server on port 552
```

Time / Timers

n= group (0/1) *There are two groups of two timer channels*
x = timer (0/1)
m = watchdog (0-5)

ms	(n1 --)	pause for "n" milliseconds.	forth
MS-TICKS	(-- n1)	Time since start in milliseconds	forth
alarm@	(t -- f)	Get alarm value	timers
alarm!	(lo hi t –)	Set alarm value	timers
autoreload!	(v t --)	Set timer t to … ?	timers
divider!	(n t --)	Timer divider 2 - 65535	timers
edgeint!	(f t --)	Edge trigger	timers
enable!	(v t --)	Timer enable/disable	timers
increase!	(v t --)	Timer increasing/decreasing	timers

int-enable!	(f t --)	Enable/disable interrupt	timers
interval	(xt usec t --)	Setup timer t to call execution token xt after usec delay	timers
levelint!	(v t --)	Level trigger	timers
onalarm	(xt t --)	Set callback	timers
t>nx	(t -- n x)	x=1 if bit0 of t=1, else x=0 n=1 if bit1 of t=1, else n=0	timers
timer!	(lo hi t --)	Set timer counter current value	timers
timer@	(t -- lo hi)	Get timer counter current value	timers
TIMG_BASE	(-- $3ff5f000)	constant	timers
TIMGn			timers
TIMGn_RTCCALICFG_REG	(n -- a)		timers
TIMGn_RTCCALICFG1_REG	(n -- a)		timers
TIMGn_Tx	(n x -- a)		timers
TIMGn_Tx_INT_CLR_REG	(n -- a)		timers
TIMGn_Tx_INT_ENA_REG	(n -- a)		timers
TIMGn_Tx_INT_RAW_REG	(n -- a)		timers
TIMGn_Tx_INT_ST_REG	(n -- a)		timers
TIMGn_Tx_WDTCONFIGm_REG	(n m -- a)		timers
TIMGn_Tx_WDTFEED_REG	(n -- a)		timers
TIMGn_Tx_WDTWPROTECT_REG	(n -- a)		timers
TIMGn_TxALARMLOHI_REG	(n x -- a)		timers
TIMGn_TxCONFIG_REG	(n x -- a)		timers
TIMGn_TxLOAD_REG	(n x -- a)		timers
TIMGn_TxLOADLOHI	(n x -- a)		timers

_REG			
TIMGn_TxLOHI_REG	(n x -- a)		timers
TIMGn_TxUPDATE_REG	(n x -- a)		timers

Interval example (--) Demonstrates starting a timed word with interval

```
timers
variable x
: hi   1 x +! ;              \ add 1 to x
' hi 1000000 0 interval      \ run hi every second
: test
begin
      x ? cr      \ display x
      100 ms      \ wait 0.1s
      key?
until ;           \ exit loop if key pressed
test
\ note the input terminal remains reponsive whilst the timer is counting down
\ there are a total of only four timer channels - so turn to multitasking if more is needed
```

Two Wire Interface / I2C

Wire.available	(-- f)	Returns the number of bytes available for retrieval with Wire.read. This should be called on a master device after a call to Wire.requestFrom	Wire
Wire.begin	(sda scl -- f)	Initiate the Wire library and join the I2C bus as a master with two GPIO pins set to act as sda and scl. This should normally be called only once.	Wire
Wire.beginTransmission	(n --)	Begin a transmission to the slave device at address n. Subsequently, queue bytes for transmission with the Wire.write function and transmit them by calling Wire.endTransmission	Wire
Wire.endTransmission	(sendstop -- f)	Ends a transmission to a slave device that was begun by Wire.beginTransmission and transmits the bytes that were queued by Wire.write Sends a stop message if sendstop=true	Wire
Wire.flush	(--)	Releases the I2C bus	Wire
Wire.getClock	(-- frequency)	Read the clock frequency set by Wire.setClock	Wire
Wire.getTimeout	(-- ms)	Gets the timeout in ms for I2C communication.	Wire

Wire.peek	(-- ch)	Reads a byte from a previously addressed slave device by using the Wire.requestFrom word. This is the same as a Wire.read except that the received-data-buffer-pointer is not incremented.	Wire
Wire.read	(-- ch)	Reads a byte that was transmitted from a slave device to a master after a call to Wire.requestFrom or was transmitted from a master to a slave	Wire
Wire.requestFrom	(address quantity sendstop -- n)	Used to request bytes from a slave device. The bytes may then be retrieved with the Wire.available and Wire.read functions. A stop message is sent after the request if sendstop is true	Wire
Wire.setClock	(frequency --)	Modifies the clock frequency for I2C communication. I2C slave devices have no minimum working clock frequency, however 100KHz is usually the baseline	Wire
Wire.setTimeout	(ms --)	Modifies the timeout in ms for I2C communication. The timeout is expressed in ms. The default value is 50 ms.	Wire
Wire.write	(a n -- n)	Writes data from a slave device in response to a request from a master, or queues bytes for transmission from a master to slave device (in-between calls to Wire.beginTransmission and Wire.endTransmission) a=start address of data, n=number of bytes	Wire

I2C device scanner (--) Displays detected I2C devices

```
wire                          \ activate the wire vocabulary
                              \ start the I2C interface using pin 21 and 22 on ESP32 DEVKIT V1
21 22 wire.begin drop         \ with 21 used as sda and 22 as scl.

: spaces ( n -- )
   for
      space
   next
  ;

: .## ( n -- )
   <# # # #> type
  ;
                                    \ not all bitpatterns are valid 7bit i2c addresses
: Wire.7bitaddr? ( a -- f )
   dup $07 >=
   swap $77 <=  and
  ;

: Wire.detect   ( -- )
   base @ >r hex
   cr
   ."      00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f"
   $80 $00 do
      i $0f and 0=
      if
         cr i .##  ."  : "
      then
      i Wire.7bitaddr? if
         i  Wire.beginTransmission
         -1 Wire.endTransmission 0 =
         if
            i .## space
         else
            ." -- "
         then
      else
         2 spaces
      then
   loop
   cr r> base !
  ;
```

Vectored Execution

defer	("vectorname" --)	Define a deferred execution vector e.g. defer myemit	forth
is	(--)	Set the vector of a deferred word e.g. ' emit is myemit - sets the deferred word myemit to execute emit when called	forth

Vocabulary

}transfer		transfer the words enclosed in curly brackets to the current library e.g. { word1 word2 word3 ... }transfer	forth
also	(--)	Duplicate the vocabulary at the top of the vocabulary stack	forth
context	(-- a)	an area to specify vocabulary search order - defaults to forth. context @ puts the current vocab id on the stack context @ @ puts the xt of the last word defined in the current vocab	forth
current	(-- addr)	points to a vocabulary thread to which new definitions are to be added	forth
definitions	(--)	Make the context vocabulary the current vocabulary	forth
forth	(--)	Make the forth vocabulary the current vocabulary	forth
internals		Make the internals vocabulary the current vocabulary	forth
interrupts		Make the interrupts vocabulary the current vocabulary	forth
ledc		Make the ledc vocabulary the current vocabulary	forth
only	(--)	Reset context stack to one item, the FORTH dictionary	forth
order	(–)	Print the vocabulary search order	forth
registers	(--)	set the current vocabulary to registers	forth
rtos		Make the rtos vocabulary the current vocabulary	forth
SD_MMC		Make the SD_MMC vocabulary the current vocabulary	forth
sealed	(--)	Alter the last vocabulary defined so it doesn't chain	forth
Serial		Make the Serial vocabulary the current vocabulary	forth
sockets		Make the sockets vocabulary the current vocabulary	forth
SPIFFS		Make the SPIFFS vocabulary the current vocabulary	forth
streams	(--)	set the current vocabulary to streams	forth
tasks		make tasks vocabulary the current one	forth

telnetd		Make the telnetdvocabulary the current vocabulary	forth
timers	(--)	Make the timers vocabulary the current vocabulary	forth
vocabulary	("name" --)	Create a vocabulary with the current vocabulary as parent	forth
web-interface		Make the web-interface vocabulary the current vocabulary	forth
WebServer		Make the WebServer vocabulary the current vocabulary	forth
WiFi		Make the WiFi vocabulary the current vocabulary	forth
Wire	(--)	Make the Wire vocabulary the current vocabulary	forth
previous	(--)	Drop the vocabulary at the top of the vocabulary stack	forth

Visual

‘edit’ is a rudimentaty ‘spiffs’ file editor suitable for editing small files – very useful.

edit	(“filename”–)	ANSI terminal file editor e.g. visual edit /spiffs/autoexec.fs	visual

The following terminal keystrokes are recognised:-

Key strokes:	Action:
Ctrl-S	Save now
Ctrl-X / Ctrl-Q	Quit, asking Y/N to save
Ctrl-L	Redraw the screen
Backspace	Delete a character backwards
Arrow keys	Move the cursor up, down, left right
PgUp /PgDown	Scroll up / down a page

Web Interface

do-serve			*web-interface*
handle-index			*web-interface*
handle-input			*web-interface*
handle1			*web-interface*
index-html			*web-interface*
index-html#	(– 2268)	constant	*web-interface*
input-stream	(– a)	stream of size 200	*web-interface*
out-size	(– 2000)	constant	*web-interface*
out-string		block of storage, size out-size+1	*web-interface*
output-stream	(– a)	stream of size out-size	*web-interface*
serve-key	(-- n)		*web-interface*
serve-type	(a n --)		*web-interface*
server	(port --)		*web-interface*
webserver	(– a)	variable	*web-interface*
webserver-task		multitasker task	*web-interface*
webui	(network-z password-z --)	login and start webui e.g. z" NETWORK-NAME" z" PASSWORD" webui	forth

WiFi

login	(network-z password-z --)	login to wifi only e.g. z" NETWORK-NAME" z" PASSWORD" login	forth
WIFI_MODE_AP	(-- 2)	access point mode: stations can connect to the ESP32 e.g. WIFI_MODE_AP WiFi.mode	WiFi
WIFI_MODE_APSTA	(-- 3)	access point and a station connected to another access point e.g. WIFI_MODE_APSTA WiFi.mode	WiFi
WIFI_MODE_NULL	(-- 0)		WiFi
WIFI_MODE_STA	(-- 1)	station mode: the ESP32 connects to an access point e.g WIFI_MODE_STA WiFi.mode	WiFi
WiFi.begin	(ssid-z password-z --)	Initializes the WiFi library's network settings and provides the current status. e.g. z" mySSID" z" myPASSWORD" WiFi.begin	WiFi
WiFi.config	(ip dns gateway subnet --)	Allows you to configure a static IP address as well as change the DNS, gateway, and subnet addresses on the WiFi shield. Packaged a.b.c.d little-endian	WiFi
WiFi.disconnect	(--)	Disconnects the WiFi shield from the current network	WiFi
WiFi.getTxPower	(-- powerx4)	Get power x4	WiFi
WiFi.localIP	(-- ip)	Get local IP	WiFi
WiFi.macAddress	(a --)	Gets the MAC Address of your ESP32 WiFi port	WiFi
WiFi.mode	(mode --)	Set WiFi mode example below	WiFi
WiFi.setTxPower	(powerx4 --)	Set power x4	WiFi
WiFi.softAP	(ssid password/0 -- success)	Software enabled Access Point – essentially behaviour like a Router	WiFi
WiFi.softAPIP	(-- ip)	Return IP address of the soft access point’s network interface.	WiFi
WiFi.softAPBroadcastIP	(-- ip)	Function to get the AP IPv4 broadcast address	WiFi

WiFi.softAPConfig	(localip gateway subnet -- success)	Configure the soft access point's network interface.	WiFi
WiFi.softAPdisconnect	(wifioff -- success)	Disconnect stations from the network established by the soft-AP.	WiFi
WiFi.softAPgetStationNum	(-- num)	Get the count of the stations that are connected to the soft-AP interface.	WiFi
WiFi.softAPNetworkID	(– id)	Get the softAP network ID	WiFi
WiFi.status	(-- n)	Returns the connection status	WiFi

WiFi.mode (mode –) Set Wifi mode

0 WIFI_MODE_NULL	In this mode, the internal data struct is not allocated to the station and the AP, while both the station and AP interfaces are not initialized for RX/TX Wi-Fi data. Generally, this mode is used for Sniffer, or when you only want to stop both the STA and the AP to unload the whole Wi-Fi driver.
1 WIFI_MODE_STA	Station mode: in this mode, will init the internal station data, while the station's interface is ready for the RX and TX Wi-Fi data.
2 WIFI_MODE_AP	AP mode: in this mode, init the internal AP data, while the AP's interface is ready for RX/TX Wi-Fi data. Then, the Wi-Fi driver starts broad-casting beacons, and the AP is ready to get connected to other stations.
3 WIFI_MODE_APSTA	Station-AP coexistence mode: in this mode, will simultaneously init both the station and the AP.This is done in station mode and AP mode. Please note that the channel of the external AP, which the ESP Station is connected to, has higher priority over the ESP AP channel.

WiFi.status (n –) Returns the connection status – n can take the following values:-

255	WL_NO_SHIELD - no WiFi shield is present
0	WL_IDLE_STATUS - WiFi.begin is called and remains active until the number of attempts expires (resulting in WL_CONNECT_FAILED) or a connection is established (resulting in WL_CONNECTED)
1	WL_NO_SSID_AVAIL - no SSID are available
2	WL_SCAN_COMPLETED - scan networks is completed
3	WL_CONNECTED - connected to a WiFi network
4	WL_CONNECT_FAILED - connection fails for all the attempts
5	WL_CONNECTION_LOST - connection is lost
6	WL_DISCONNECTED - disconnected from a network

WiFi connection example (--) Connect and disconnect from WiFi demo web-interface also WiFi

```
: status.    ( -- )                                  \ print WiFi connection status
." Current WiFi status = " WiFi.status . cr
;

: test       ( -- )
WIFI_MODE_STA WiFi.mode                              \ set to connect to an access point
z" yourroutername" z" yourpassword" WiFi.begin       \ attempt to connect
3000 ms
status.                                              \ report our Wifi link status
." Your assigned IP address = " WiFi.localIP
ip. cr                                               \ report local IP address
WiFi.disconnect                                      \ disconnect
." Now disconnecting" cr
3000 ms
status.                                              \ report WiFi status again
;

\ N.B. edit the above with your router's name and password!!
```

Word definition

,	(n --)	store a value into the dictionary space	forth
;	(--)	stop compiler, and finish word definition e.g. : gday ." good day to you" ;	forth
:	("wordname " --)	start compiler mode, creates a word definition e.g. : hi ." hello world" ;	forth
:noname	(– xt)	Create a word with no name, leaving it's execution token on the stack. The xt would then usually be stored elsewhere from which the word can be executed	forth
'	("wordname " -- xt)	xt = execution token of the word that follows in the input stream e.g. ' words puts 1073654684 on the stack. Errors if word not found, stopping execution	forth
[	(--)	stop compiling the input stream and start executing - sets state=true	forth
[']	(-- xt)	xt = execution of the word that follows inside a : definition e.g. : COMING ['] HELLO 'aloha ! ;	forth
]	(--)	Stop executing and start compiling the input stream, sets state=false	forth
{	(--)	Mark the start of a local variable block	forth

align			forth
aligned	(addr1 -- addr2)	converts an address on the stack to the next higher cell boundary, to help accessing memory by cells	forth
allot	(n –)	Allocate n bytes for storage and increment HERE by that space. See also the word ALLOCATE	forth
c,	(c --)	compile byte c at the next available location in the word definition	forth
constant	(n "name" --)	create a constant whose name follws in the input stream, value n. e.g. 12 constant dozen	forth
CREATE	(-- ; -- pfa)	create an empty dictionary entry <name>, returns the parameter field address when executed	forth
DOES>	(-- addr)	Used with create in defining new defining words e.g. : array (n -- ; i -- addr) \ new array type variable create cells allot does> swap cells + ; 10 array baba \ create a 10 cell array named baba 0 baba puts the 1st element address on the stack 1 baba puts the 2nd element address on the stack 10 0 baba ! stores 10 in the 1st element.	forth
IMMEDIATE		Marks the last defined word as immediate - it will execute immediately if called whilst compiling a word	forth
literal	(n -- ; -- n)	add top of stack into the word being compiled at the next free memory location. When the word is run, place n top of stack	forth
postpone	("text" --)	Skip leading space delimiters. Parse name delimited by a space. Find name. Append the compilation semantics of name to the current definition. Useful when an immediate word needs to be compiled in a word definition instead of immediately executing. Use instead of the obsolete COMPILE or [COMPILE] , you don’t have to remember if word is immediate or normal	forth
recurse	(--)	Allows a word to call itself e.g. : FACTORIAL DUP 2 < IF DROP 1 EXIT THEN DUP 1- RECURSE * ; so 5 FACTORIAL leaves 120 on the stack	forth

SMUDGE	(--)	stops the current word being defined being found during a dictionary lookup	forth
to	(n “valuename” –)	e.g. 24 to myvalue - sets myvalue = 24	forth
+to	(n “valuename” –)	e.g. 4 +to myvalue – set myvalue = myvalue+4	forth
value	(n "valname" -- ; -- n)	creates a value, named with the word that follows in the input stream, initialised n	forth
value-bind	(xt-val xt –)		forth
variable	("varname" -- ; -- addr)	variable takes the next word in the input stream as the name and reserves space for a variable	forth

Local variables (n1 n2 --) Demonstrates defining a word where the input parameters on the stack are labelled for enhanced readability

```
: summ  { foo bar }              \ bar is topmost element of the stack
       foo bar + .               \ add the two top values on the stack and display the result
;

\ so 2 3 summ results in 5 displayed
```

:NONAME example (–) Demonstrates creating words with no names, which can nevertheless be executed by execution token - saves space in the dictionary if the word is never used by name

```
:noname ." Saturday" ;
:noname ." Friday" ;
:noname ." Thursday" ;
:noname ." Wednesday" ;
:noname ." Tuesday" ;
:noname ." Monday" ;
:noname ." Sunday" ;

create (day) ( --- addr)              \ an array of execution tokens for the 7 headerless words
, , , , , , ,

: day. ( n -- )                        \ valid n=0-6
cells (day) + @ execute ;

\ executing 2 day. displays Tuesday etc.
```

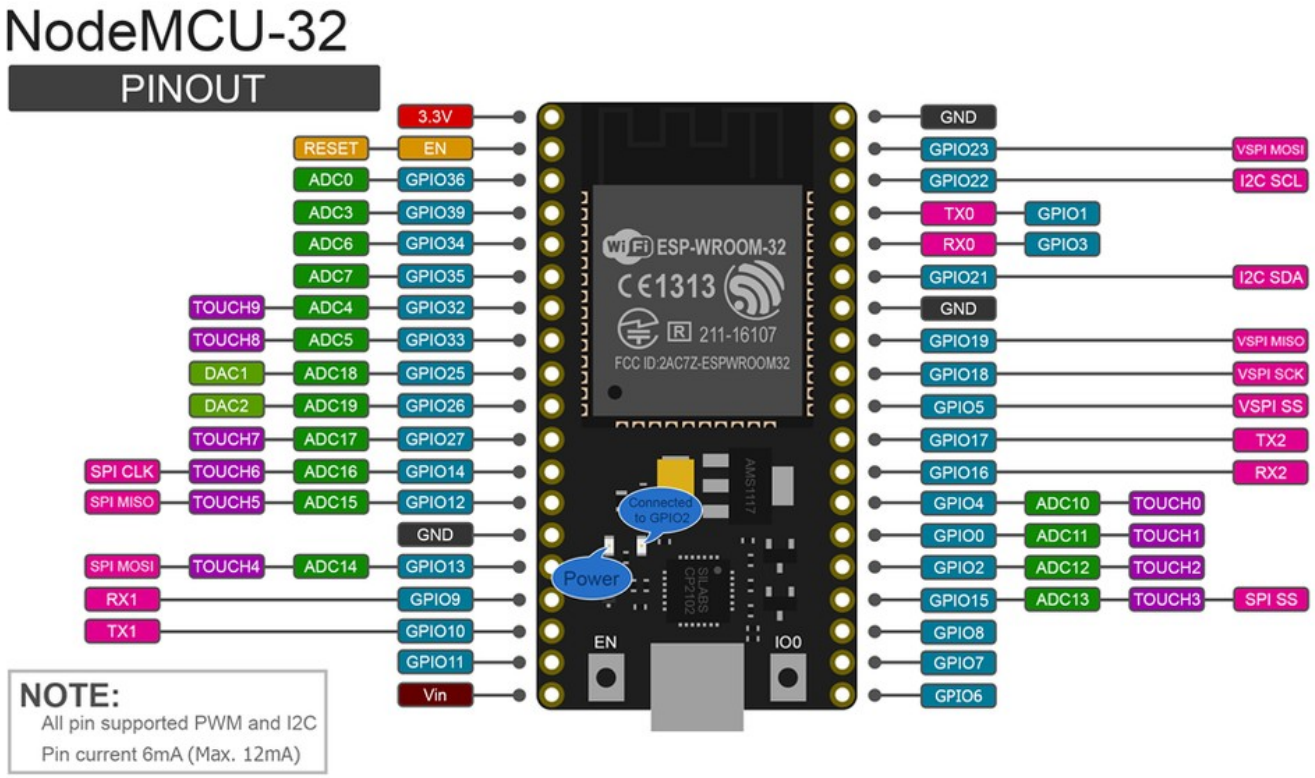
Useful documentation

Bradley Nelsons's ESP32forth home page, which includes downloads & **installation instructions**
Marc PetreMann's excellent webpage on ESP32forth programming
Forth2020 on facebook
ESP32 module buying guide
The latest version of this document is found here
Introduction to Xtensa assembly language here, here and here
Xtensa Instruction Set Architecture

Useful Code

The Forth2020 group's archive of examples is very useful
Marc Petremann's ESP32forth github page
The github ESP32forth / forth2020group code page
My code on github

NodeMCU-32 pin-out



This module has two LED lamps, one indicates power on, the other is driven from GPIO2. ESP32forth will turn the GPIO2 LED on at boot up.

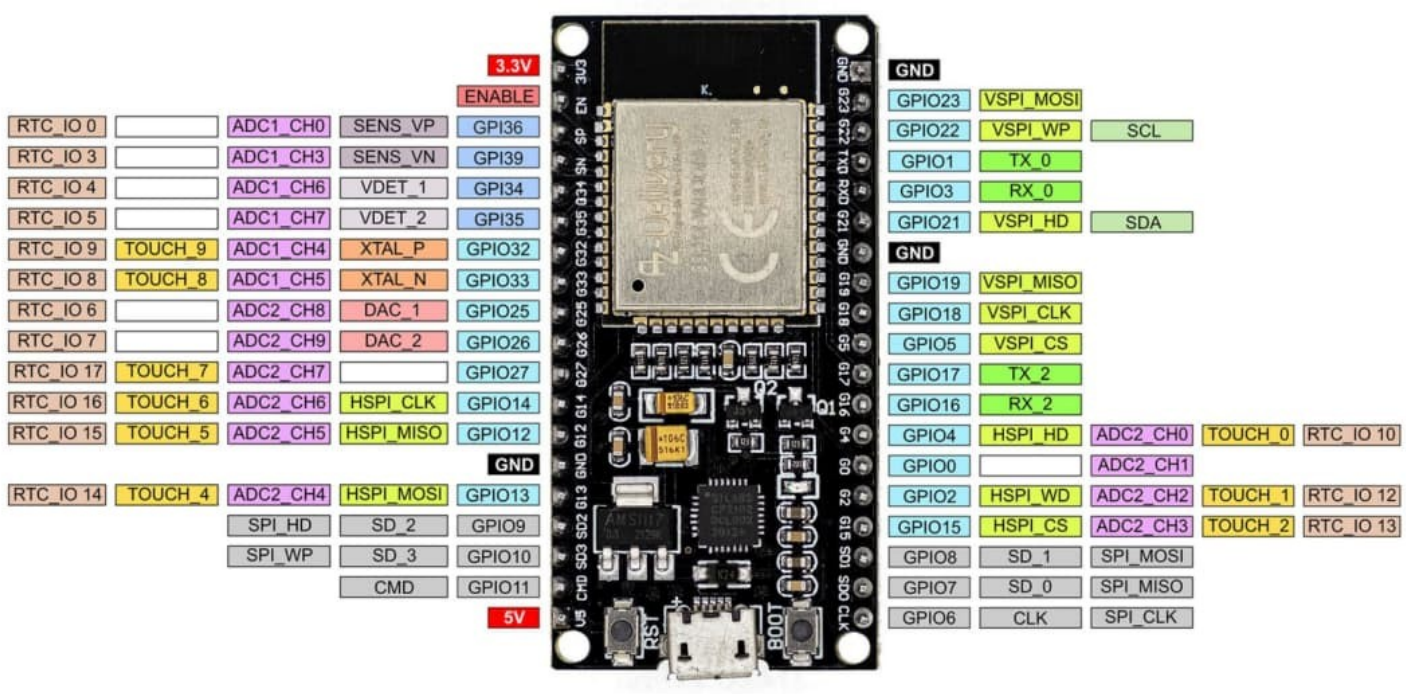
The BOOT button is connected to GPIO0 and can be used by user programs.

ESP32 WROOM 32 pin-out

This is a module I bought which is often found on Ebay and supports ESP32forth:-

The module has one blue LED, which turns on when the ESP32 outputs characters to the terminal via the USB port. It does not light up when characters are being received by the ESP32. The blue LED cannot be lit up under GPIO control.

The BOOT button is connected to GPIO0 and can be used by user programs.



Conclusion

This glossary was first compiled in 2022 by Bob Edwards, retired EMC engineer in SW U.K. He always has it open on screen, when he programs ESP32forth and hopes you'll find it useful too.

Appendix 1 Internal Words

The following words are system internal ‘worker words’ and are liable to change from version to version of ESP32forth. For that reason they are non-preferred for User programs – caveat emptor. Nevertheless, when nothing else will do – here they are:-

Block File system

arduino-default-use	(--)	attempt to open block file "/spiffs/blocks.fb"	internals
block-data	(-- addr)	A 1024 byte buffer for handling block file data	internals
block-dirty	(-- n)	value, default=0	internals
clobber	(a --)	fill one block of 1024 characters with spaces	internals
clobber-line	(a -- a')	fill one block file line with spaces	internals
common-default-use	(--)	attempt to open block file "blocks.fb"	internals
grow-blocks	(n --)	Increase the size of the file by n blocks	internals

Block File Editor

e'	(n --)	used internally by e	internals

Branching

0BRANCH			internals
BRANCH			internals
[SKIP]		deferred word, defaults to [SKIP]', used internally in [ELSE]	internals
[SKIP]'		used internally in [ELSE]	internals

Character I/O

?arrow.	(--)	If system variable arrow is true, the user prompt of an arrow followed by the data stack contents shows the system is ready for user input	internals
?echo	(c --)	If system variable echo=true, then display the ascii character c, else if echo=false, c is dropped	internals
***emit**	(n –)		internals
***key**	(– n)		internals
dump-line	(a –)	print address line leaving room – part of the	internals

		dump word	
eat-till-cr			internals
input-buffer	(-- addr)	character buffer, size input-limit	internals
input-limit	(-- 200)	constant - the maximum permitted input char count before being terminated with <cr>	internals
line-pos	(– n)	value,	internals
line-width	(-- n)	value, defaults to 75 = the number of characters per line on the output stream	internals

Debug

raw.s	(--)	display the data stack on one line of the display	internals

Files

arduino-remember-filename	(-- addr cnt)	returns the filename "/spiffs/myforth"	internals
dirname	(a n --)		internals
ends/	(a n -- f)		internals
include-file	(fh --)		internals
include+			internals
included-files		value, default = 0	internals
path-join	{ a a# b b# -- a n }		internals
raw-included	(a n --)		internals
remember-filename	(-- a n)	Deferred word specifying the platform specific default snapshot filename - defaults to arduino-remember-filename = "/spiffs/myforth"	internals
restore-name	("name" --)	Restore a snapshot from a file	internals
save-name	(a n --)	Save a snapshot of the current vocabulary to a file	internals
sourcedirname	(-- a n)		internals
sourcefilename	(-- a n)		internals
sourcefilename!	(a n --)		internals
sourcefilename&		value, default =0	internals
sourcefilename#		value, default =0	internals
starts../	(a n -- f)		internals
starts./	(a n -- f)		internals

Internal words

'cold	(– a)		internals
(local)	(addr n --)		internals
)leaving			internals
@line	(n --)		internals
ALITERAL			internals
default			
default-remember-filename	(– string)	default value is “myforth”	
DOLIT			internals
evaluate-buffer			internals
EVALUATE1			internals
exit=	(xt -- f)	f=true if the execution token at top of stack is that of 'exit', else f=false	internals
leaving	(-- addr)	variable	internals
leaving,			internals
leaving(			internals
notfound	(addr cnt n --)		internals
onlines	(n xt -- n xt)		internals
park-forth	(-- addr)		internals
park-heap	(-- addr)		internals
parse-quote	(-- addr cnt)		internals
RAW-YIELD	(--)		internals
restore-name	(“name” –)		
save-name			
saving-base			internals
scope	(-- addr)	variable	internals
scope-clear		reset the r stack and set scope=0	internals
scope-create	(a n --)		internals
scope-depth	(-- addr)	variable	internals
scope-doer		creates a new word of type scope-doer	internals
scope-template		is an instance of a scope-doer	internals
see-all			internals
see-loop			internals

see-one	(xt -- xt+1)		internals
see-xt	(xt --)		internals
see.	(xt --)		internals
setup-saving-base	(–)		
tib-setup			internals
use?!			internals
voc-stack-end	(-- addr)		internals
voc.	(voc --)		internals
xt-find&	(xt -- xt&)		internals
xt-hide	(xt --)		internals
xt-transfer	(xt --)		internals

Memory

‘heap-size			internals
‘heap-start			internals
ca@	(a – n)		internals
fill32			internals
heap_caps_free			internals
heap_caps_malloc			internals
heap_caps_realloc			internals
LONG-SIZE	(– 4)	Returns the size of a 32 bit long in bytes	internals
MALLOC	(n -- a | 0)	System malloc - reserve n bytes of memory, returns start address a if successful, else returns 0 on failure	internals
MALLOC_CAP_32BIT	(-- 2)	constant	internals
MALLOC_CAP_8BIT	(-- 4)	constant	internals
MALLOC_CAP_DEFAULT	(-- 4096)	constant	internals
MALLOC_CAP_DMA	(-- 8)	constant	internals
MALLOC_CAP_EXEC	(-- 1)	constant	internals
MALLOC_CAP_INTERNAL	(-- 2048)	constant	internals
MALLOC_CAP_IRAM_8BIT	(-- 8192)	constant	internals
MALLOC_CAP_PID	(n1 -- n1 n2)	n1 range is 3-28, then n2 = 32*2^^ (n1-3) e.g. if n1=3, n2=32; if n1=4. n2=64 etc	internals

MALLOC_CAP_RETENTION	(-- 8192)	constant	internals
MALLOC_CAP_SPIRAM	(-- 1024)	constant	internals
mem=	(a1 a2 n -- f)	f=true if the memory contents at a1 is equal to that at a2 for n bytes	internals
REALLOC	(a n -- a | 0)	System realloc	internals
SYSFREE	(a --)	System free - release memory previously reserved with MALLOC?	internals

Number I/O

#f+s	(r –)		internals
digit	(u -- c)	converts an integer to an ascii char e.g. 5 is converted to 53	internals

Serial Communication

serial-key	(-- c)	reads the next character from the serial port	internals
serial-key?	(-- f)	an alias for Serial.available	internals
serial-type	(addr count --)	send the counted string at addr to the serial port	internals

Stack functions

?stack	(–)	throws an error if stack underflow or overflow has occurred	internals
'stack-cells			internals

String functions

'tib			internals
$@			internals
$place	(addr cnt --)		internals
S>NUMBER?	(addr cnt -- n f=true | f=false)	converts the counted ascii string stored at addr to number n with f=true, else just returns f=0, no n	internals

System

'context	(-- addr)	system variable	internals
'heap	(-- addr)	system variable	internals
'notfound	(-- addr)	system variable – stores execution token of the word to be called in the event a word is not found in the dictionary. This normally displays ERROR: <word> NOT FOUND!	internals
'SYS	(-- addr)	system variable - used as the base address for system variables e.g. : 'context 'SYS 7 cells + ;	internals
‘boot			internals
‘boot-size			internals
‘cold	(– addr)	Address of the word that will run on start-up	internals
arrow	(-- addr)	variable, default=true if so, the user will be prompted by --> to show the forth system is ready for user input	internals
autoexec	(--)	at system start, check for autoexec.fs on the flash drive and run if present. **N.B. the filename must have been saved lowercase else it won’t be recognised and your program won’t autostart**	internals
default-remember-filename	(-- addr cnt)	returns the string myforth	internals
esp32-bye	(–)	Restarts ESP32forth as though from switch-on	internals
esp32-stats	(–)	Displays chip model, type, clock speed, number of cores, flash chip size, system heap stats	internals
free.	(nf nu --)	Part of the sign-on message printed by raw-ok	internals
growth-gap	(-- $4000	constant	internals
raw-ok	(–)	Sign-on message – displays version, clockrate, number of cores, space, dictionary status, stack info	internals

Tasks – multitasking

YIELD			internals
yield-step		Assigned to yield-task	internals
yield-task			internals

Vocabulary

‘context			internals
‘latestxt			internals
‘notfound			internals
>vocnext	(xt -- xt)		internals
forth-wordlist	(-- n)	constant, defined as current @	internals
last-vocabulary	(– n)		internals
latestext	(– xt)		internals
nonvoc?	(xt – f)		internals
see-all	(–)		internals
see-vocabulary	(voc)		internals
size-all	(–)		internals
size-vocabulary	(voc)		internals
voc.	(voc –)		internals
vocs.	(voc --)		internals
voclist	(–)	Display all vocabularies	internals
voclist-from	(voc –)		internals

Word definition

-TAB	(– 64)	constant	internals
}?	(addr cnt --)	used internally by { in defining local variables	internals
+TAB	(– 32)	constant	internals
BUILTIN_FORK	(– 4)	constant	internals
DOCOL			internals
DOCON			internals
DOCREATE			internals
DODOES			internals
DOSET			internals
DOVAR			internals
IMMEDIATE_MARK	(– 1)	constant	internals
immediate?	(xt -- f)	f=true if the word whose execution token is on the stack is an immediate word e.g. ' if immediate? returns true ' load immediate? returns false	internals

MARK	(– 128)	constant	internals
NONAMED	(– 16)	constant	internals
SMUDGE	(– 2)	constant	internals

Orphan Words

‘argc			internals
‘argv			internals
‘runner			internals
RAW-YIELD			internals

Appendix 2 Notes on using the Sockets Dictionary

A very useful guide to programming with Internet Sockets is located here.

Addresses

IPV4 internet addresses are represented by a‘sockaddr’ structure in memory:-

	← 32 bit cell →	
long1	[len][family][port]	len = 16, the number of bytes in the structure family = 2, AF_INET for internet use
long2	[address]	
long3	[unused]	
long4	[unused]	

A sockaddr can be created like a variable with a user provided name e.g. sockaddr data_in

len and family fields are set automatically.

All 16 bit fields and upwards are ‘big-endian’ i.e. The ms byte of the number resides at the lowest memory address. Since ESP32forth is ‘little-endian’ a number of transfer words are available to read and write fields in sockaddr structures:-

->addr!	(n a --)	set big-endian address in sockaddr	sockets
->addr@	(a -- n)	get big-endian address from sockaddr	sockets
->port!	(n a --)	set big-endian port in sockaddr	sockets
->port@	(a -- n)	get big-endian port from sockaddr	sockets

Making a Transmission Control Protocol (TCP) Connection

When making connections a ‘server’ is the name given to the semi-permanent entity which can be conncted to by one or more less-permanent ‘clients’. These clients can disconnect from the server when done. e.g. A thermometer server in your greenhouse may be connected to by a client running on your PC in the house. Before shutting down the PC, the client application disconnects from the thermometer. It’s all a bit like making a telephone call: Dial, Accept Call,Chat, Hangup.

The tcp server has to execute the following steps in sequence:-

gethostbyname	(hostnamez -- hostent/0)	Look up the host by name, using a zero terminated string. Returns a pointer to a hostent structure or 0 if failed
->h_addr	(hostent – a)	Get host address from hostent (returned by gethostbyname)
socket	(domain type protocol – sock/err)	Make a socket

bind	(sock addr addrlen -- 0/err)	A server will bind to an address
listen	(sock backlog -- 0/err)	Listen for socket connections and limit the queue of incoming connections - 'backlog' is the max no. of connections that can be put on hold. If you only want to ever allow one connection, set backlog = 0
sockaccept	(sock1 addr addrlen -- sock2/err)	The sockaccept function extracts the first connection on the queue of pending connections, creates a new socket sock2 with the same socket type protocol and address family as the specified socket, and returns a new file descriptor for sock2 in sockaddr addr, addrlen. N.B. make sure here that addrlen is an address of variable, not a number! It must be set to 16 (sockaddr size in bytes) before calling sockaccept.

Once the connection is established, the client and server can repeatedly exchange data with:-

send	(sock a n flags -- n/err)	send data at address a, n bytes long
recv	(sock a n1 flags -- n2/err)	receive data to buffer at address a, required size n1. returns number of bytes actually read n2. It's up to you to call recv again until all the data is read

After all communication is done, the connection is closed with:-

CLOSE-FILE	(sock -- ior)	close the socket

Let's look at the forth programming needed for a tcp client:-

Get the host address

```
z" google.com" gethostbyname constant google.com      \ look up the host details using the name
google.com →h_addr                                    \ from the details extract the host ip addr
.ip 142.251.46.238 ok                                 \  .ip converts and displays the address
```

Create a sockaddr and populate it with address + port

```
sockaddr googleaddr                                  \ create a sockaddr

80 googleaddr →port!                                 \ save the port address required

google.com →h_addr googleaddr ->addr!                \ get host address and save in
googleaddr
```

Create a Socket

```
AF_INET SOCK_STREAM 0 socket value sock              \ create ‘sock’ SOCK_STREAM = TCP
```

Connect to the server

```
sock googleaddr sizeof(sockaddr_in) connect throw    \ throw used because connect may fail
```

Send an HTTP request

```
S” GET / HTTP/1.0” sock write-file throw             \ send a string using write-file

: semit (chr sock – ) swap >r rp@ swap 1 swap write-file throw rdrop ;

: semit (chr sock – ) swap >r rp@ 1 0 send 0< throw rdrop;      \ either version will send a
character

: scr 13 sock semit 10 sock semit ;                  \ send CR-LF

scr scr
```

Read part of the reply

```
here 100000 sock read-file throw constant len        \ read the response from google.com

here len type                                        \ using read-file
```

Close the connection

```
sock close-file throw
```

Another Example - Reading and Writing blocks of bytes

Another example which demonstrates sending data between two ESP32s is shown here, including instructions on how to run the demo.

Communication by User Datagram Protocol (UDP)

If TCP communication was like making a connection with a telephone, UDP is a bit like radio broadcasting – there's no end-to-end formal connection to be made. Consequently data can be missed, out of sequence or damaged.

In pseudo-code, the client has to execute the following steps:-

- gethostbyname (hostname)
- socket (domain,type,protocol)
- bind (sock,addr,addrlen)
- sendto (sock,data, datalen, flags, addr, addrlen)
- recvfrom (sock,data,datalen,flags,addr. *addrlen)
- close (sock)

Some example forth:-

Create a 'listening' address

sockaddr incoming

9999 incoming ->port!

Create a socket and bind to the address

AF_INET SOCK_DGRAM 0 socket value sockfd

sockfd non-block throw

sockfd incoming sizeof(sockaddr_in) bind throw

Read an incoming packet

sockaddr received

variable received-len

sizeof(sockaddr_in) received-len !

sockfd msg len 0 received

received-len recvfrom to len

received ->addr@ ip. ." :" received ->port@ .

space space msg swap type cr

Appendix 3 Catch and Throw

This article was copied from CATCH and THROW (turboforth.net) – it's not my work.

Another useful article on CATCH and THROW appears here. The words CATCH and THROW, discussed in this section, provide a method for propagating error handling to any desired level in an application program. THROW may be thought of as a multi-level EXIT from a definition, with CATCH marking the location to which the THROW returns. Suppose that, at some point, word A calls word B, whose execution may cause an error to occur. Instead of just executing word B's name, word A calls word B using the word CATCH. Someplace in word B's definition (or in words that B's definition may call) there is at least one instance of the word THROW, which is executed if an error occurs, leaving a numerical throw code identifier on the stack. After word B has executed and program execution returns to word A just beyond the CATCH, the throw code is available on the stack to assist word A in resolving the error. If the THROW was not executed, the top stack item after the CATCH is zero.

THROW (errcode –)

When you detect an error in your program, you can use THROW to "throw an error". In practice, THROW is used with a number (on the stack) so that the type of error can be identified. Let's take an imaginary word, DIV which divides two numbers. You want to check for a division by zero, and if so, act upon it. First, the Forth 83 way:

```
: DIV ( quotient divisor -- result ) dup 0= abort" DIV: Divide by 0 error." / ;
```

That's about the best you can do in Forth 83 without having to resort to passing flags to indicate if the division succeeded or not. The problem is that in the event of 0 being passed, the running program will stop. That's what ABORT and ABORT" does. Even worse, DIV can't tell us which word passed 0 to DIV in the first place, so it's not particularly useful.

Let's look at how we would trap errors using THROW :-

```
: DIV ( quotient divisor -- result ) dup 0= if 99 throw else / then ;
```

Here, if the divisor is 0, we "throw" error code 99 (which, in our program, means "divide by zero" - I chose 99 at random - it could be any value you like). But to where do we "throw" this 99? Or put it another way, who, or what is going to catch this error?

Best to throw an error at the deepest convenient part of the program. Using THROW without CATCH is OK during development because the system has a CATCH which leads to the word Error being displayed. Just barely useful for debugging.

CATCH (xt – 0 | errorcode)

CATCH will catch an error thrown by THROW. The critical difference is, this allows your program to gracefully handle the error situation (prompt the user to change disks if the disk is full, rather than just abort, causing the user to lose his magnus opus in your word processor application). Let's have a look at how we would use CATCH with our DIV example above :-

```
: test-div ( quotient divisor -- result )
  ['] div catch dup 0<> if
    dup 99 = if
```

```
        ." Divide by zero error"
      else
        throw
      then
    then ;
```

The stack signature for CATCH is as follows:

```
CATCH ( ... xt -- 0|error_code )
```

What this means is, CATCH expects the execution token (xt) of the word you want to execute, in our example, DIV, to be on the stack. CATCH itself will then execute that word on your behalf. After DIV executes, CATCH can determine if control came back to CATCH via THROW or by a normal termination of the word. If the word terminated normally, CATCH puts a 0 on the stack (meaning that CATCH did not catch anything). If control came back to CATCH via THROW, the THROW code will be on the stack.

Best to catch at the highest point convenient in the program, some say

Thusly, in our example test above we test the error code returned by CATCH. If it's 0 then DIV did not throw anything, everything worked. If the return code is not zero however, something went wrong in DIV. We then examine the code, and, if it's 99 we indicate a divide by zero error.

If the error code is not 99 (which is the only thing that test-div is interested in) then something else went wrong (maybe the word / threw a different error of its own). All we know is, we're interested in error codes 0 and 99, and if aint either of them then it's "sombody else's problem". In this case, we can THROW the error again, which will cause it to be caught by the next higher CATCH in the chain (if there is one).

Another THROW – CATCH example:-

```
: 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 ;
```

Appendix 4 Starting a program automatically on switch on

Method 1

The system will automatically attempt to mount SPIFFS filesystem at /spiffs. It will then, at start-up, attempt to load /spiffs/autoexec.fs

One way this feature can be used to configure the Web UI to start by default. When doing this, be sure to test your Web UI settings work well first.

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

To remove a previously configured autoexec.fs you will need to be able to reboot in a mode with Forth. One way to do this is to search for the line in the .ino file that refers to autoexec.fs and replace it with a different name. Then run the following:

s" /spiffs/autoexec.fs" delete-file

Method 2

Snap-shotting the dictionary may not be stable across reinstallations of the C build of Forth.

A collection of non-standard words is provided that allow snap-shotting the dictionary and restoring it at startup, with a start word.

SAVE ("name" --) Saves a snapshot of the current dictionary to a file.

RESTORE ("name" --) Restore a snapshot from a file.

REMEMBER (--) Save a snapshot to the default file

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

STARTUP: ("name" --) Save a snapshot to the default file arranging for

"name" to be run on startup.

REVIVE (--) Restore the default filename.

RESET (--) Delete the default filename.

Here's an example usage:

: welcome ." Hello!" cr 100 0 do i . loop cr ;

startup: welcome

bye

(Next boot will run a custom startup message)

reset

(Reset removes the custom message)

The INTERNALS vocabulary has some additional words for more control.

SAVE-NAME (a n --) Save a snapshot if the current vocabulary to a file.

RESTORE-NAME (a n --) Restore a snapshot from a file.

'COLD (-- a) Address of the word that will be run on startup.

REMEMBER-FILENAME (-- a n) Deferred word specifying the platform specific
default snapshot filename.

Appendix 5 Spotting coding errors using the data stack

Errors in long compiles

When uploading source code via the terminal to ESP32forth, it's not easy to spot any ERROR message before it quickly disappears off the top of the screen. Has the compilation been fault free or not, you ask yourself? Here's a dodge that makes it obvious:-

Before uploading your source code, put a number on the data stack.

--> 1234

ok

1234 →

Then compile the source code. If the code compiles without error, that number will still be present on the stack after it finishes. If there has been an error, the stack will have been emptied and the number will no longer be displayed as part of the user prompt. Now scroll the terminal display back to where the number reappears in the prompt. This is where the first ERROR in your code was reported.

Errors in stack usage

When testing code, put (say) 3 numbers on the data stack:-

--> 1 2 3

ok

1 2 3 →

Whilst running the code under test, check these three numbers remain intact and are not accidentally being chewed up.