ANSI Standard Forth—Quick Reference

Commands That Can Help You Start or End Work Sessions

CORE	**ENVIRONMENT?**	strAddr u -- x flag strAddr u -- ... flag strAddr u -- 0(unmatched)	*L*eave descriptive values and a nonzero flag when environment has string at n1. So, n3 is an int with a (max) size for "/counted-string", "/hold", "/pad", "address-unit-bits", "return-stack-cells", and "stack-cells"; a max value for "max-char", "max-d", "max-n", "max-u", and "max-ud"; or a true flag for "core" and "core-ext" if the full wordset is honored.
CORE EXT	**UNUSED**	-- u	*P*ush count of free dictnary address units.
CORE	**WORDS**	--	*L*ist words in first-searched wordlist.
TOOLKIT EXT	**BYE**	--	*E*xit the Forth system.
FACILITY EXT	**TIME&DATE**	-- +n +n +n +n +n +n	*P*ush hour, min, sec, day, mo, yr.

Commands to Inspect Memory, Debug, & View Code

CORE	**DEPTH**	-- +n	*P*ush count of stack cells in use.
BLOCK EXT	**LIST**	u --	*T*ype the contents of disk block n1.
TOOLKIT	**DUMP**	addr u --	*D*ump data in n2 memory locations.
TOOLKIT	**?**	~addr --	*D*isplay integer stored at cell addr n1.
TOOLKIT	**.S**	... -- <same>	*D*isplay stack contents, leaving it intact.
TOOLKIT	**SEE** *defName*	--	*D*ecompile the named routine.

Commands That Change Compilation & Interpretation Settings

CORE	**BASE**	-- ~addr	*P*ush cell addr with the number base.
CORE	**DECIMAL**	--	*M*ake 10 the current number base.
CORE* [1]	**FORGET** *defName*	--	*D*iscard *defName* and any definition since.
CORE EXT	**HEX**	--	*M*ake 16 the current number base.
CORE EXT	**MARKER** *name*	--	*D*efine *name* as an op that **FORGET**s itself.
SEARCH EXT	**ALSO**	--	*D*uplicate the first-searched wordlist.
SEARCH	**DEFINITIONS**	--	*A*ppend new defs to first-searched wordlist.
SEARCH EXT	**FORTH**	--	*S*earch the Forth wordlist first.
SEARCH	**FORTH-WORDLIST**	-- selx	*P*ush the wordlist ID for Forth.
SEARCH	**GET-CURRENT**	-- selx	*P*ush the wordlist ID receiving new defs.
SEARCH	**GET-ORDER**	-- selx... n	*P*ush wordlists in search order, depth.
SEARCH EXT	**ONLY**	--	*M*inimize the search-order.
SEARCH EXT	**ORDER**	--	*P*rint the wordlist search order.
SEARCH EXT	**PREVIOUS**	--	*P*op the first-searched wordlist.
SEARCH	**SET-CURRENT**	selx --	*M*ake n1 the wordlist receiving new defs.
SEARCH	**SET-ORDER**	selx... n --	*M*ake n-deep wordlists the search order.
SEARCH	**WORDLIST** *name*	--	*D*efine the *name* wordlist.
	name	-- selx	*P*ush *name* 's assoc wordlist ID.
TOOLKIT EXT	**ASSEMBLER**	--	*S*earch assembler wordlist first.
TOOLKIT EXT	**EDITOR**	--	*S*earch editor wordlist first.

Source Code Preprocessing, Interpreting, & Auditing Commands

CORE EXT	**.(** *abc...)*	--	*P*rint chars up to close paren.
FILE	**INCLUDE-FILE**	... selx -- ...	*I*nterpret an opened file with ID selx.
FILE	**INCLUDED**	... strAddr u -- ...	*O*pen file n2/n3 names, interpret file.
BLOCK	**LOAD**	... u -- ...	*I*nterpret disk block n2 until exhausted, then restore the original input-stream.
BLOCK EXT	**THRU**	... u u -- ...	*L*ike **LOAD**, using block range n2 to n3.
TOOLKIT EXT	**[IF]** *wordInumber...* ¬	flag --	*E*xecute words between **[IF]**-**[ELSE]** or **[IF]**-**[THEN]** if n1 is true; else execute words beyond **[ELSE]** or **[THEN]**.
TOOLKIT EXT	**[ELSE]** *wordInumber...* ¬	--	
TOOLKIT EXT	**[THEN]**	--	

Comment-Introducing Operations

CORE EXT [4]	**** *abc...<newline>*	--	*S*tart a comment spanning the line.
CORE [2]	**(** *abc...)*	--	*S*tart a parenthetical comment.

Dynamic Memory Operations

MEMORY	**ALLOCATE**	u -- ~addr 0 u -- ~addr x(failure)	*A*llocate n1 addr units starting at n2 and leave zero flag if successful.
MEMORY	**FREE**	~addr -- 0 ~addr -- x(failure)	*D*eallocate range of memory alloc'd by **ALLOCATE**, leave zero if successful.
MEMORY	**RESIZE**	~addr u -- ~addr' 0 ~addr u -- ~addr x(failure)	*R*eallocate a memory range sized to n2 addr units, leave zero if successful.

String Operations (see also "More Input/Output Operations")

CORE EXT*	**CONVERT**	ud strAddr -- ud strAddr	*S*uperceded by **>NUMBER**.
CORE	**COUNT**	strAddr -- strAddr u	*P*ush length of counted string at n1, incr address n1 to first char of string.
CORE EXT	**ERASE**	addr u --	*C*lear bits of n2 char addrs beg at n1.
CORE	**FILL**	strAddr u char --	*S*et string at addr n1 to all n3 chars.

CORE	**HOLD**	char --	*P*repend n1 to number string. See **<#**.
CORE	**MOVE**	addr addr u --	*C*opy n3 vals at addr n2 to addr n3.
CORE	**>NUMBER**	ud strAddr u -- ud' strAddr u'	*A*dd number string n2/n3's val to n1 to get n4. Leave str n5 of length n6 with any unconverted chars. Uses **BASE**.
CORE	**<#** *prependWord...*	¬ --	*S*tart ud-to-string prepending sequence.
CORE	**#>**	ud -- strAddr	*D*rop dividend n1, push ref to assoc str.
CORE	**#**	ud -- ud'	*E*xtract next digit from n1, prepend it.
CORE	**#S**	ud -- ud(zero)	*E*xtract signif digits left in n1, prepend.
CORE	**SIGN**	n --	*P*repend minus to number str if n1<0.
STRING	**BLANK**	strAddr u --	*C*onvert n2 leading chars of n1 to spaces.
STRING	**CMOVE**	strAddr strAddr u --	*L*eft-to-right copy n3 chars to n2.
STRING	**CMOVE>**	strAddr strAddr u --	*S*ame as **CMOVE**, but right-to-left copy.
STRING	**COMPARE**	strAddr u strAddr u -- 0 | -1 | 1	*L*eave 0 if n1/n2 matches n3/n4.
STRING	**SEARCH**	strAddr u strAddr u -- addr u flag	*L*eave true flag if n1/n2 found in n3/n4, preceded by unscanned chars, preceded by location of first such char (n5).
STRING	**/STRING**	strAddr u n -- strAddr u	*L*eave n1/n2 with n3 fewer leading chars.
STRING	**-TRAILING**	strAddr u -- strAddr u	*D*ecrement n2 by n1/n2's pad spaces.
FLOAT	**>FLOAT**	strAddr u -- float flag strAddr u -- 0(failure)	*C*onvert n2 chars of string at n1 to a floating-point value, leave result flag.
FLOAT	**REPRESENT**	float strAddr u -- n flag flag float strAddr u -- n 0(not-neg) flag float strAddr u -- n flag 0(failure)	*C*onvert float n1 to a mantissa string at n2 with n3 signif digits. Leave exp n4, sign-flag n5, in-range flag n6.

[redacted] Disk Input/Output Operations Using Files or Block Buffers

BLOCK	**BLOCK**	u -- ~addr	*A*ssign a blk buffer as the current block, read disk blk n1 into it, leave assoc addr.
BLOCK	**BUFFER**	u -- ~addr	*A*cts like **BLOCK**, but doesn't read disk.
BLOCK EXT	**EMPTY-BUFFERS**	--	*F*ree all disk buffers, incl those updated.
BLOCK	**FLUSH**	--	*S*ave updated buffers, then free all bufs.
BLOCK	**SAVE-BUFFERS**	--	*S*ave changes, but do not free buffers.
BLOCK EXT	**SCR**	-- ~addr	*P*ush cell addr with last-**LIST**ed block.
BLOCK	**UPDATE**	--	*M*ark current blk buffer as updated.
FILE	**BIN**	x -- x'	*M*odify file mode n1 to select binary.
FILE	**CLOSE-FILE**	selx -- 0|x(failure)	*C*lose file ID n1, leave result flag.
FILE	**CREATE-FILE**	strAddr u x -- selx 0 strAddr u x -- x x(failure)	*C*reate a file of name at n1/n2, open it in n3 mode, leave file ID and result flag.
FILE	**DELETE-FILE**	strAddr u -- 0|x(failure)	*D*elete file refd by n1/n2,leave result flg.
FILE	**FILE-POSITION**	selx -- ud 0|x(failure)	*L*eave cur pos n2 for file ID n1, res flag.
FILE	**FILE-SIZE**	selx -- ud 0|x(failure)	*L*eave size of file ID n1, result flag.
FILE EXT	**FILE-STATUS**	strAddr u -- x 0|x(failure)	*L*eave file n1/n2 status, 0 flag if exists.
FILE EXT	**FLUSH-FILE**	selx -- 0|x(failure)	*F*lush buffers for file ID n1, leave flag.
FILE	**OPEN-FILE**	strAddr u x -- selx 0 strAddr u x -- x x(failure)	*O*pen file of name n1/n2 in n3 mode. Leave file ID n4, result flag n5.
FILE	**R/O**	-- x	*P*ush read-only file mode flag.
FILE	**R/W**	-- x	*P*ush read-write file mode flag.
FILE	**READ-FILE**	strAddr u selx -- u 0 strAddr u selx -- u x(failure)	*O*pen file with file ID n3, reading n2 or fewer chars into string at n1; leave count of chars read (n4), result flag.
FILE	**READ-LINE**	strAddr u selx -- u x 0|x(failure) strAddr u selx -- u 0(eof) 0	*L*ike above, n4 chars are moved to string n1. If at eof, leave 0 beneath 0 result flag.
FILE EXT	**RENAME-FILE**	strAddr u strAddr u -- 0|x(failure)	*R*eset filename to that given by n3/n4.
FILE	**REPOSITION-FILE**	ud selx -- 0|x(failure)	*S*et position n1 for file refd by ID n2.
FILE	**RESIZE-FILE**	ud selx -- 0|x(failure)	*S*et size n1 for file refd by file ID n2.
FILE	**W/O**	-- x	*P*ush write-only file mode flag.
FILE	**WRITE-FILE**	strAddr u selx -- 0|x(failure)	*W*rite n2 chars at n1 to file ID n3.
FILE	**WRITE-LINE**	strAddr u selx -- 0|x(failure)	*W*rite n2 chars at n1, adding newline.

[redacted] More Input/Output Operations

CORE	**ACCEPT** *abc...<newline>*	strAddr +n -- +n(count-recd)	*R*ead line of up to n2 chars, store at n1.
CORE	**CR**	--	*E*mit newline character or equiv seqence.
CORE	**.** **D.** (DOUBLE)	n --	*P*rint value n1 according to **BASE**.
CORE EXT	**.R** **D.R** (DOUBLE)	n n --	*A*s above, right-adjusted in n2-wide field.
CORE	**."** *abc...*"	--	*C*ompile string and string-print op.
CORE	**EMIT**	x --	*E*mit the printable char assoc with n1.
CORE EXT*	**EXPECT** *abc...<newline>*	strAddr +n --	*R*ead up to n2 chars, store at n1.
CORE	**KEY** *<keypress>*	-- char	*R*ead one keyboard char.
CORE	**SPACE**	--	*E*mit space character.
CORE	**SPACES**	n --	*E*mit n1 space characters.
CORE	**TYPE**	strAddr u --	*P*rint first n2 chars of string at n1.
CORE	**U.**	u --	*P*rint n1 as an unsigned value (See **.**).
CORE EXT	**U.R**	u n --	*P*rint n1 like **.R**, but as an unsigned value.
FLOAT EXT	**F.**	float --	*P*rint n1 using fixed-point notation.
FLOAT EXT	**FE.**	float --	*P*rint n1 in exponential notation.
FLOAT EXT	**FS.**	float --	*P*rint n1 in scientific notation.
FACILITY	**AT-XY**	u u --	*S*et **EMIT**'s dest to column n1, row n2.
FACILITY EXT	**EKEY** *<keyboard event>*	-- selx	*R*eceive a keyboard event.
FACILITY EXT	**EKEY>CHAR**	selx -- char flag selx -- selx 0(failure)	*C*onvert keyboard event n1 to char if possible, else push false flag.
FACILITY EXT	**?EKEY**	-- flag	*P*ush 0 if no keyboard event to handle.
FACILITY EXT	**?EMIT**	-- flag	*P*ush 0 if display device is unavailable.

FACILITY	**?KEY**		-- flag	*P*ush 0 if no keypress is available.
FACILITY EXT	**MS**		u --	*W*ait at least n1 milliseconds.
FACILITY	**PAGE**		--	*M*ake **EMIT**'s dest a new page/screen.

Arithmetic and Logical Operations

CORE	**ABS**	**DABS** (DOUBLE)	n -- u	*L*eave absolute value: | n1 |
CORE	**AND**		x x -- x	*B*itwise logical and: n1 AND n2
CORE	**FM/MOD**		d n -- n n	*L*eave (n1 % n2), (n1 / n2). Floored div.
CORE	**INVERT**		x -- x	*I*nvert all the bits of n1.
CORE	**LSHIFT**		x u -- x	*L*eft-shift the bits of n1 n2 times.
CORE	**M***		n n -- d	*M*ultiply: n1 * n2
CORE	**MAX**	**DMAX** (DOUBLE)	n n -- n	*D*rop lower of n1 and n2.
CORE	**MIN**	**DMIN** (DOUBLE)	n n -- n	*D*rop higher of n1 and n2.
CORE	**-**	**D-** (DOUBLE)	nlu nlu -- nlu	*S*ubtract: n1 – n2
CORE	**MOD**		n n -- n	*L*eave remainder: n1 % n2
CORE	***/MOD**		n n n -- n n	*L*eave (n1*n2 % n3), (n1*n2 / n3).
CORE	**/MOD**		n n -- n n	*L*eave (n1 % n2), (n1 / n2).
CORE	**NEGATE**	**DNEGATE** (DOUBLE)	n -- n	*N*egate n1: 0 – n1
CORE	**1+**		n -- n	*A*dd one: n1 + 1
CORE	**1-**		n -- n	*S*ubtract one: n1 – 1
CORE	**OR**		x x -- x	*B*itwise logical or: n1 OR n2
CORE	**+**	**D+** (DOUBLE)	nlu nlu -- nlu	*A*dd: n1 + n2
		M+ (DOUBLE)	dlud n -- dlud	
CORE	**+!**	**D+!** (DOUBLE)	nlu ~addr --	*A*dd n1 to value at n2.
CORE	**RSHIFT**		x u -- x	*R*ight-shift the bits of n1 n2 times.
CORE	**/**	**D/** (DOUBLE)	n n -- n	*D*ivide: n1 / n2
CORE	**SM/REM**		d n -- n n	*L*eave (n1 % n2), (n1 / n2). Symm div.
CORE	*****		nlu nlu -- nlu	*M*ultiply: n1 * n2
CORE	***/**	**M*/** (DOUBLE)	n n n -- n	*L*eave n1 * n2 / n3.
CORE	**2***	**D2*** (DOUBLE)	x -- x	*S*hift 1 bit, 0 placed in least signif bit.
CORE	**2/**	**D2/** (DOUBLE)	x -- x	*S*hift 1 bit, unchanged most signif bit.
CORE	**UM***		u u -- ud	*M*ultiply: n1 * n2
CORE	**UM/MOD**		ud u -- u u	*L*eave (n1 % n2), (n1 / n2)
CORE	**XOR**		x -- x	*B*itwise exclusive-or: n1 XOR n2
FLOAT	**F***		float float -- float	*M*ultiply: n1 * n2
FLOAT	**F/**		float float -- float	*D*ivide: n1 / n2
FLOAT	**F+**		float float -- float	*A*dd: n1 + n2
FLOAT	**F+!**		float ~addr --	*A*dd n1 to value at n2.
FLOAT	**FLOOR**		float -- float	*R*ound n1 (floored).
FLOAT	**FMAX**		float float -- float	*D*rop lower of n1 and n2.
FLOAT	**FMIN**		float float -- float	*D*rop higher of n1 and n2.
FLOAT	**FNEGATE**		float -- float	*N*egate n1: 0 –n1
FLOAT	**FROUND**		float -- float	*R*ound n1 (unfloored).

Number-Type Conversion Operations

CORE	**S>D**	n -- d	*C*hange integer to double (sign extnd).
DOUBLE	**D>S**	d -- n	*C*hange double to signed integer.
FLOAT	**D>F**	d -- float	*C*hange signed double to float.
FLOAT	**F>D**	float -- d	*C*hange float to signed double.

Commands to Define Data Structures

CORE	**CONSTANT** *name*	x --	*D*efine *name* to push n1, a value
	name	-- x	that should not be subject to change.
CORE EXT	**VALUE** *name*	--	*D*efine *name* to push the value last
	name	x --	stored into it with **TO**. See **TO**.
CORE	**VARIABLE** *name*	--	*D*efine *name* to push an address in
	name	-- ~addr	data space suitable for storing and fetching cell-size vals. See **@** and **!**.
DOUBLE	**2CONSTANT** *name*	x x --	*D*efine *name* to push n1/n2, values
	name	-- x x	that should not be subject to change.
DOUBLE	**2VARIABLE** *name*	--	*D*efine *name* to push an address
	name	-- ~addr	in data space suitable for storing and fetching a double. See **2@** and **2!**.
FLOAT	**FCONSTANT** *name*	float --	*D*efine *name* to push n1, a value that
	name	-- float	should not be subject to change.
FLOAT	**FVARIABLE** *name*	--	*D*efine *name* to push an address in
	name	-- ~addr	data space suitable for storing and fetching a float. See **F@** and **F!**.

Memory-Stack Transfer Operations

CORE	**C@**	~addr -- char	*F*etch char n2 from aligned address n1.
CORE	**C!**	char ~addr --	*S*tore char n1 at aligned address n2.
CORE	**@**	~addr -- x	*F*etch n2 from aligned cell address n1.
CORE	**2@**	~addr -- x x	*F*etch n2/n3 from aligned address n1.
CORE	**!**	x ~addr --	*S*tore n1 at aligned cell address n2.
CORE	**2!**	x x ~addr --	*S*tore n1/n2 at aligned cell address n3.
CORE EXT	**TO** *defValueName*	x --	*S*tore n1 in named **VALUE** variable.
FLOAT	**F@**	~addr -- float	*F*etch n2 from aligned float address n1.
FLOAT	**F!**	float ~addr --	*S*tore n1 at aligned float address n2.
LOCAL	**TO** *defLocalName*	x --	*S*tore n1 in named **LOCAL** variable. See **(LOCAL)**, **LOCALS|**.

Comparison Operations

CORE	**=**	**D=** (DOUBLE)	x x -- flag	*L*eave true if n1 is equal to n2.
CORE	**>**		n n -- flag	*L*eave true if n1 is greater than n2.
CORE	**<**	**D<** (DOUBLE)	nlu nlu -- flag	*L*eave true if n1 is less than n2.
CORE EXT	**<>**		nlu -- flag	*L*eave true if n1 is not equal to n2.
CORE	**U<**	**DU<** (DOUBLE EXT)	u u -- flag	*L*eave true if n1 is less than n2.
CORE EXT	**U>**		u u -- flag	*L*eave true if n1 is greater than n2.
CORE EXT	**WITHIN**		nlu nlu nlu -- flag	*L*eave true if (n1≤n2<n3) or (n1≥n2>n3).
CORE	**0=**	**D0=** (DOUBLE)	x -- flag	*L*eave true if n1 is equal to zero.
CORE EXT	**0>**		n -- flag	*L*eave true if n1 is greater than zero.
CORE	**0<**	**D0<** (DOUBLE)	n -- flag	*L*eave true if n1 is less than zero.
CORE EXT	**0<>**		x -- flag	*L*eave true if n1 is not equal to zero.
FLOAT	**F<**		float float -- flag	*L*eave true if n1 is less than n2.
FLOAT	**F0=**		float -- flag	*L*eave true if n1 is equal to zero.
FLOAT	**F0<**		float -- flag	*L*eave true if n1 is less than zero.

System Constants & Facilities for Generating ASCII Values

CORE	**BL**	-- char	*P*ush ASCII code for space character.
CORE	**CHAR** *abc...<space>*	-- char	*P*ush value of first char of next word.
CORE	**[CHAR]** *abc...<space>*	-- char(Runtime)	*C*ompile *a* as ASCII and op to push it.
CORE EXT	**FALSE**	-- flag	*P*ush false flag.
CORE EXT	**TRUE**	-- flag	*P*ush true flag.

Forming Definite Loops (Compiling-Mode Only)

CORE /EXT	**DO**| **?DO**	¬	nlu nlu --	*M*ark start of block run 1 or more times.
	word|number...	¬		**?DO** skips block if n2 ≥ n1 to begin.
CORE	**I**	¬	-- nlu	*P*ush **DO**'s new n2 value this iteration.
CORE	**J**	¬	-- nlu	*L*ike **I**, but outer do-loop's new n2 value.
CORE /CORE	**LEAVE|UNLOOP**	¬	--	*L*eave loop early. **LEAVE** skips words
	word|number...	¬		through nearest **LOOP** or **+LOOP**.
CORE	**LOOP**|		--	*C*ontinue back at **DO** or **?DO** if after
CORE	**+LOOP**		n --	incrementing **DO**'s orig n2 each cycle,
				n2 < n1. **+LOOP** uses n1 as increment.

Forming Indefinite Loops (Compiling-Mode Only)

CORE	**BEGIN**	¬	--	*M*ark the beginning of a list of words
	word|number...	¬		subject to repeated execution.
CORE	**AGAIN**|	¬	--	*I*terate back to **BEGIN** always.
CORE	**UNTIL**|	¬	flag --	*I*terate back to **BEGIN** until true n1.
CORE	**WHILE**	¬	flag --	*E*xecute words through **REPEAT**, then
	word|number....	¬		back again to **BEGIN** until false n1; then
CORE	**REPEAT**		--	skip words through **REPEAT**.

More Facilities for Defining Routines (Typically Compiling-Mode Only)

CORE [3]	**ABORT**		x... --	*E*mpty data stack, perform **QUIT**.
CORE [3]	**ABORT"** *abc..."*		x... x --	*C*ompile string up to dbl quote and
				op to display it and **ABORT** if n2≠0.
CORE EXT	**C"** *abc..."*		-- strAddr	*C*ompile string delimited by dbl quote
				& op that pushes its addr at run time.
CORE EXT	**CASE**	¬	--	*M*ark beginning of case block formed
	word|number...	¬		with the following 3 types of words.
CORE EXT	**OF**	¬	x x --	*E*xecute words through **ENDOF** if n2 = n1
	word|number...	¬		(resuming after **ENDCASE**). If n2≠n1,
				resumes after next **ENDOF**.
CORE EXT	**ENDOF**	¬	--	*M*ark the end of previous **OF**'s
	word|number...	¬		scope.
CORE EXT	**ENDCASE**		x --	*D*rops n1. Follows default-case code.
CORE	**:** *name* |	¬	--	(:) *S*tarts routine definition for *name*.
CORE EXT	**:NONAME**	¬	-- xtoken	**:NONAME** starts nameless routine def,
	word|number...	¬		leaving n1 for use with **EXECUTE**.
CORE	**;**	¬	--	Finishes defining *name* to run any
	IMMEDIATE			compiled *word|number* references.
				IMMEDIATE will make *name* a compiler
				extension (directive). See **IMMEDIATE**.
CORE	**EXIT**		--	*L*ets routine exit at loc other than end.
CORE	**IF** *word|number...*	¬	flag --	*E*xecute words between **IF-ELSE** or
CORE	**ELSE** *word|number...*	¬	--	**IF-THEN** if n1 is true; else execute words
CORE	**THEN**		--	beyond **ELSE** or **THEN**.
CORE	**[** *word2interpret...*		--	*S*tarts interpreting following words.
CORE	**QUIT**		--	*E*mpty return stack, reset I/O to
				console, and repeatedly: read line,
				interpret it, display system prompt.
CORE	**RECURSE**		--	*C*ompiles self-reference in definition.
CORE	**]** *word2compile...*		--	*S*tarts compiling non-immediate words.
CORE [2]	**S"** *abc..."*		-- strAddr	*C*ompile string delimited by dbl quote &
				op that pushes its address at run time.
EXCEPTION	**CATCH**		... xtoken -- ... 0	*R*un execution thread for xtoken,
			... xtoken -- ... n(exceptn)	saving its execution frame parameters.

EXCEPTION	**THROW**	0 --		*D*rop n1 and continue normally if 0.
		–1 --		If n1 is –1, call **ABORT**.
		–2 --		If n1 is –2, perform **ABORT"**.
		n -- n		Else, exit to latest **CATCH** execution frame, adjusting input stream and stacks to associated settings.
TOOLKIT EXT	**CODE** *name*	--		*S*tart *name*'s machine-code definition.
	codeWord...			*codeWord*s are nonstandard words.
TOOLKIT EXT	**;CODE** *codeWord...*	--		*S*witch to compiling a routine definition with machine-code-compiling words.
LOCAL	**(LOCAL)**	strAddr +n --		*N*onzero n2 causes identifier at n1 to be
		strAddr 0 --		defined as a value-pushing local. See **TO**.
LOCAL EXT	**LOCALS|** *name...* **|**	x... --		*C*reate as many local variables as names
	name	-- x		before |, initialized to n stack values. Normal limit is 8. See **TO**.

Manipulating Stack Items

CORE	**DROP**	x --		*P*op n1.
CORE	**2DROP**	x x --		*P*op n1 and n2.
CORE	**DUP**	x -- x x		*P*ush a copy of n1.
CORE	**2DUP**	x x -- x x x x		*P*ush copies of n1 and n2.
CORE	**?DUP**	x -- x x		*P*ush a copy of a nonzero n1. Else
		0 -- 0		leave stack unchanged.
CORE EXT	**NIP**	x x' -- x'		*R*emove n1.
CORE	**OVER**	x x' -- x x' x		*P*ush a copy of n1.
CORE	**2OVER**	x x x' x' -- x x x' x' x x		*P*ush copies of n1 and n2.
CORE EXT	**PICK**	x... +n -- x... x		*L*eave copy of input item n2+2 deep.
CORE EXT	**2>R**	x x --		*M*ove n1 and n2 to return stack.
CORE	**R>**	-- x		*M*ove n1 from return to data stack.
CORE EXT	**2R>**	-- x x		*M*ove n1/n2 from return to data stack.
CORE	**R@**	-- x		*C*opy val atop return stack to data stack.
CORE EXT	**2R@**	-- x x		*C*opy n1/n2 from return to data stack.
CORE EXT	**ROLL**	x... +n -- x...		*R*otate to TOS the input item n2+2 deep.
CORE	**ROT**	x x' x'' -- x' x'' x		*R*eposition n1 to top of stack.
CORE	**SWAP**	x x' -- x' x		*E*xchange positions of n1 and n2.
CORE	**2SWAP**	x x x' x' -- x' x' x x		*E*xchange positions of n1/n2 and n3/n4.
CORE EXT	**TUCK**	x x' -- x' x x'		*C*opy top item into 3rd stack slot.
DOUBLE EXT	**2ROT**	x x x' x' x'' x'' -- x' x' x'' x'' x x		*M*ove n1/n2 to top of stack.
FLOAT	**FDEPTH**	-- +n		*F*or float stack use. See **DEPTH**.
FLOAT	**FDROP**	float --		*D*rop n1 from top of float or data stack.
FLOAT	**FDUP**	float -- float float		*D*uplicate n1 on float or data stack.
FLOAT	**FOVER**	float float' -- float float' float		*P*ush copy of n1 on float or data stack.
FLOAT	**FROT**	float float' float'' -- float' float'' float		*R*epositions n1 to top of float/data stack.
FLOAT	**FSWAP**	float float' -- float' float		*E*xchange n1 and n2 on float/data stack.

Constructing Compiler and Interpreter System Extensions

CORE	**ALIGN**	**FALIGN** (FLOAT)	--		*A*lign the data-space pointer.
CORE	**ALIGNED**	**FALIGNED** (FLOAT)	addr -- ~addr		*A*lign address to store a data type.
CORE	**ALLOT**		n --		*A*dd n1 to data-space pointer.
CORE	**>BODY**		xtoken -- ~addr		*R*eplace n1 with ref'd word's data addr.
CORE	**C,**		char --		*S*tore n1 in data space for latest def.
CORE	**CELL+**	**FLOAT+** (FLOAT)	~addr -- ~addr		*I*ncrement n1 to next cell address.
CORE	**CELLS**	**FLOATS** (FLOAT)	n -- n		*P*ush address units reqd for n1 cells.
CORE	**CHAR+**		~addr -- ~addr		*I*ncrement n1 to next char addr.
CORE	**CHARS**		n -- n		*P*ush address units reqd for n1 chars.
CORE	**,**		n|u --		*S*tore n1 in data space for latest def.
CORE EXT	**COMPILE,**		xtoken --		*C*ompile word assoc with n1 into def.
CORE EXT	**[COMPILE]** *immWord*		--		*C*ompile compiler-extending actions of named immediate word into current def.
CORE	**CREATE** *name*		--		*D*efines a data word (*name*) that
	name		-- ~addr		pushes n1, the data-space pointer's
	: *newDefiner*	¬			value upon *name*'s creation. The sequence
	word|number...	¬			*newDefiner name* creates a similar
	CREATE	¬			data-word instance by executing
	word|number...	¬			*newDefiner*'s code up to **DOES>**.
CORE	**DOES>**	¬	-- ~addr		Such a word's data address (n1) will be
	word|number... ;				processed by any *newDefiner* code between **DOES>** and **;** (semicolon).
CORE [4]	**EVALUATE**	... strAddr u -- ...			*I*nterpret the contents of string n1/n2 as though it is the current input stream.
CORE	**EXECUTE**	... xtoken -- ...			*E*xecute the word ref'd by xtoken.
CORE	**HERE**	-- addr			*P*ush the next free loc in data-space.
CORE	**IMMEDIATE**	--			*S*et latest def to run when it appears as part of a new definition. See **;** also.
CORE	**>IN**	-- ~addr			*P*ush cell addr with input buf's cur pos.
CORE	**[']** *word*	--			*C*ompile named word's execution addr
		-- xtoken(Runtime)			(inline) and op to push it onto stack.
CORE	**LITERAL**	x --			*C*ompile n1 (inline) into current def
		-- x(Runtime)			and op to push it onto stack at run-time.

CORE EXT	**PAD**		-- strAddr	*P*ush addr of work string space.
CORE EXT	**PARSE** *abc...<delimiter>*	char	-- strAddr u	*P*arse input stream up to an n1 char.
CORE	**POSTPONE** *word*		--	*C*ompile op to compile *word*. If *word* is set immediate, acts like **[COMPILE]**.
CORE EXT	**QUERY**		--	*M*ake console the current input stream, read line into TIB, store zero in **>IN**.
CORE EXT [2,4]	**REFILL**		-- flag -- 0(failure)	*F*ill input buffer, read from console if necessary (see **QUERY**), push true flag. If a string is current input stream, do nothing beyond pushing false flag.
CORE EXT	**RESTORE-INPUT**	selx... n	-- flag	*H*onor n2-deep input stream parameters.
CORE EXT	**SAVE-INPUT**		-- selx... n	*P*ush cur input-stream parameters, depth.
CORE	**SOURCE**		-- strAddr u	*P*ush addr of input-stream buffer.
CORE EXT [2]	**SOURCE-ID**		-- 0 | -1 | x	*P*ush indication of the input stream source: console, string, or file ID.
CORE EXT*	**SPAN**		-- ~addr	*P*ush cell addr with **EXPECT**'s char count.
CORE [1]	**STATE**		-- ~addr	*P*ush cell address with system state value.
CORE EXT*	**TIB**		-- strAddr	*P*ush addr of text input buffer (TIB).
CORE EXT*	**#TIB**		-- ~addr	*P*ush cell address with TIB char count.
CORE	**'** *defName*		-- xtoken	*S*earch for named word, push assoc n1.
CORE	**WORD** *abc...<delimiter>*	char	-- strAddr	*C*opy input stream through char n1 to n2.
SEARCH	**FIND**	strAddr	-- xtoken 1(immed) -- xtoken -1 -- strAddr 0(failure)	*F*ind first match of word refd by n1 in search order wordlists. If found leave n2, immediacy-status n3. Else push zero.
SEARCH	**SEARCH-WORDLIST**	strAddr u selx strAddr u selx strAddr u selx	-- xtoken 1 -- xtoken -1 -- 0(unmatched)	*A*cts like **FIND**—but n2 is the string length and n3 is ID of wordlist to be searched exclusively.
STRING	**SLITERAL**	strAddr u	--	*C*ompile string n1/n2 (inline) into current def, & compile op to push equiv parameters onto stack at run time.
	(Runtime)		-- strAddr u	
DOUBLE	**2LITERAL**	x x	--	*C*ompile n1/n2 (inline) into cur def and op to push them onto stack at run time.
	(Runtime)		-- x x	
FLOAT	**FLITERAL**	float	--	*C*ompile n1 (inline) into cur def and op to push them onto stack at run time.
	(Runtime)		-- float	
BLOCK	**BLK**		-- ~addr	*P*ush cell addr with current disk blk no.
TOOLKIT EXT	**AHEAD**		-- orig	*P*ush branch's forward-resolving, inline data field addr onto control-flow stack.
TOOLKIT EXT	**CS-PICK** origldest... u		-- origldest... dest	*S*ee **PICK**. Apply to control-flow stack.
TOOLKIT EXT	**CS-ROLL** origldest... u		-- origldest...	*S*ee **ROLL**. Apply to control-flow stack.

Endnotes

1,2,3,4 Wordsets so marked overlap other wordsets. If your system uses an alternate wordset version of the associated word, the description and stack effect shown here may be inaccurate. The overlapping wordsets are indicated by footnote marks: [1] TOOLKIT EXT, [2] FILE, [3] EXCEPTION EXT, [4] BLOCK EXT

* An asterisk following a wordset name indicates that the associated word is considered obsolete.

Stack Notations: The notations n1, n2, and so forth refer to the values notated in the stack diagram for a word; n1 references the leftmost stack item notation, n2 the next notation, and so forth, with no distinction between input or return values, and with no intended disclosure regarding the data type of the referenced stack item. Stack diagram terms are not the same as those used in the ANSI standard. For example, ~addr is a shorthand for *aligned address*, with no indication of the data type for which it is aligned (contextual cues usually suffice). Like x, the terms selx and xtoken indicate unspecified cell types.

Sorting: Symbol-only names are sorted according to their pronounceable names (as defined in the standard). So, **2/** (two-slash) is found in the t's, **+!** (plus-store) in the p's. (However, the leading punctuation in **/MOD** is ignored, so it appears together with other words that start with m, such as **MOD**.)

Floating-Point Wordset Extensions

While the standard provides forty-one FLOAT EXT operations, only three display operations are shown here. Not shown are fourteen trigonometric operations, one comparison operation, two display-oriented variables, eight exponential operations, one absolute-value operation, six 32-bit-specific (single-float) operations, and six 64-bit-specific (double-float) operations.

Brought to you by the Forth Interest Group

A copy of this quick reference card is provided free of charge by FIG as a service to its members. FIG sponsors an annual Forth conference in California at Pacific Grove, publishes the leading journal dedicated to the Forth programming language, and carries a full line of mail-order books and disks related to Forth.

Notices

FIG makes no guarantees or promises about the accuracy of the information provided here. The ANSI X3J14 committee, through their published standard, is the definitive source of information. Please refer to the official ANS Forth document for embellishment and clarification of items. Formal comments or "requests for interpretation" should be sent to:

X3J14 c/o X3 Secretariat • 1250 Eye St. NW, Suite 200 • Washington, DC 20005-3922 • USA

For inquiries about the Forth Interest Group and any of its member services, mail or fax your inquiry to the following address:

P.O. Box 2154 • Oakland, California 94621 • (fax) 510-535-1295