Chapter 18
Forth Extensions

Introduction

This chapter contains alphabetized Forth words which are intended to supplement the standard set of Forth 83 words. The words listed in this chapter are not those that are part of the HForth or JForth glossaries, you should see the individual manual supplements for those. These are miscellaneous Forth words added mainly for use in and by HMSL. By confining oneself to the Forth 83 standard, HMSL specific words, and these extensions, code that is portable between the Macintosh and Amiga can be written.

Occasionally below, we will list a word that is defined in HForth or JForth, for emphasis, ease of reference, and so on.

Words Added to Forth for HMSL

In the stack diagrams that follow, standard Forth syntax is used. Stack at entry to the left of the dashes, stack at exit to the right. Top of the stack is always rightmost item, on either side of the "--". Spaces always separate stack items.

`$.  ( $string -- , prints a string )`

Prints the given string. Defined simply as COUNT TYPE.

`'C ( <word>  -- CFA )`

Pronounced "tick-see." Expects a word in the input stream. Gets the CFA of the word specified. This is a very high level word which gets the appropriate address on any system. It is an important word and is used quite a bit in HMSL and in user written HMSL routines.

'C is used whenever you need to put one word inside another that will in turn be executed later. 'C is the word that allows vectored execution.

It allows for host-independent calling of Forth CFA's, and returns the executable address of a word no matter what type of dictionary structure is in use in Forth. HMSL is currently supported under three such Forth's (JForth, MACH-2, and ERG/CCM 68k Forth), each with a different dictionary structure.execution.

Forth Purist's Note, `'C` **vs.** `‘`: In HMSL, there is no difference between the standard `‘` and `'C`. However, Forth purists may note that we do not conform completely to the Forth standard for the word `‘` in the word `'C`, so we added it to avoid confusion. You can use `‘` in HMSL whenever you use `'C`.

Those familiar with *Starting Forth* will realize that `'` is problematic in many different Forth systems if you try to use Brodie's examples directly. In general, it is good programming practice, even withon HMSL, to limit 'C's use within colon definitions. It is a good idea to do all initialization inside colon definitions that are called at run time just before executing your piece/application.

`3DROP  ( n1 n2 n3 -- , drops top three from the stack )`

This simple utility (2DROP DROP) is useful for instrument definitions (see Instruments chapter).

`<=  ( a b -- flag )`

Returns a flag whose value depends on whether or not *a* is less than or equal to *b*.

Examples: 3 5 <= returns true, but 5 3 <= returns false.

`>=  ( a b -- flag )`

Returns a flag whose value depends on whether or not *a* is greater than or equal to *b*. It returns false if *a* is less than *b*.

Examples: 7 5 >= returns a true flag. 5 7 >= returns a false.

?TERMINAL/64 (-- key? , returns 1 if key pressed, tests periodically)

A variant on the ?TERMINAL operation, which tests for whether or not a key has been pressed. ?TERMINAL/64 tests periodically (every 64 times) rather than constantly, which speeds up the process. Scans the terminal 1/64 as many times as ?TERMINAL would have, so less intrusive in time critical operations.

BEEP (--)

Different for Macintosh and Amiga. Standard CTRL-G implemented differently on both systems.

On Macintosh: Rings a "bell" and flashes the screen. Does this by calling SYSBEEP (Macintosh operating system utility).

On Amiga: Flashes the screen. Same as saying 7 EMIT.

Related Word: synonymous with BELL.

BELL (-- , Synonym for BEEP)

BREAK (-- , dump stack and pause)

Dumps stack and waits for character. Aborts if 'A' is hit. Useful for debugging.

CFA->LFA (cfa -- lfa)

Converts CFA to LFA. This word was written because different versions of Forth differ in how they implement the words associated with finding addresses. For consistency, we define the terms here:

CFA = address that can be passed to EXECUTE

PFA = address of data in a CREATE/DOES word.

LFA = address of backward link for dictionary.

NFA = address of count byte for word name.

CFA->NFA (cfa -- nfa)

Converts CFA to NFA.

CFA->PFA (cfa -- pfa)

Converts CFA to PFA.

CLIPTO (num low high -- n-clipped , clips num to range)

If num is lower than low, then it returns low. If num is higher than high, then it returns high. Otherwise it returns num.

Example:

```
     5 7 9 CLIPTO . ( prints 7 )
```

DEBUG.TYPE ($string -- , types string if IF-DEBUG is true)

Useful for debugging.

DECR (address --)

Decrements a variable's contents by 1. Expects the variable's address on the stack.

Related Word: INCR

DISABLE (address --)

Resets a 32 bit memory location to 0. Typically used to reset flags.

```
     COUNTER DISABLE
     COUNTER @ .  \  prints 0
```

ENABLE (address --)

Sets the contents of a variable to 1. Expects the address of the variable to be set.

ESCAPE? (-- , aborts if key hit)

Causes Forth to ABORT if a key is hit.

EVEN-UP (value -- value | value+1 , adds one if not even)

Given an even number, it returns the number. Given an odd number, it adds one to the number. The result is always even.

```
3 EVEN-UP .    \  prints 4.
```

IF-DEBUG (-- addr , variable flag for debug mode)

If this variable is set true, then HMSL will print various diagnostic messages while running. It will, for example, print messages traveling up and down the hierarchy as the polymorphous executive executes morphs.

IF-TESTING (-- addr , variable flag for test compilation)

If HMSL is compiled when this flag is true then various test procedures will be compiled along with HMSL. See .IF in the Forth manual.

INCR (address --)

Increments the contents of a variable by 1. Expects the variable's address on the stack.

Related Word: DECR

ISBLACK (char -- flag, true if black)

A "Black" character has a value between hex 20 and hex 7F.

ISDIGIT (char -- flag, true if a digit)

```
ASCII 6  ISDIGIT \  returns true
ASCII S  ISDIGIT \ returns false.
```

ISLOWER (char -- flag, true if lowercase)

ISPRINT (char -- flag , result is true if printable)

ISUPPER (char -- flag, true if uppercase)

K: (value -- , sets a constant)

Alternate for CONSTANT. Commonly used in a lot of Forths to save typing. Expects the constant's name to follow immediately in the input stream. Similar to using V: for VARIABLE.

```
50 K: UPPER
```

LFA->CFA (lfa -- cfa)

Converts LFA to CFA. See CFA->LFA.

LFA->NFA (lfa -- nfa)

Converts LFA to NFA.

MS (number-of-milliseconds -- , synonym for MSEC)

See MSEC below.

MSEC (number-of-milliseconds -- , delay)

Delays for the specified number of milliseconds. On the Macintosh, this is just some nested DO LOOPs. The accuracy of this word is controlled by a variable MSEC-DELAY that is a count for the inner loop. It is synonymous with MS. It is calibrated at compile time by the word CALIBRATE.MSEC. Watch out for this word, it is not that useful in writing demos and tests, since it completely hangs the system for the specified time.

NFA.MOVE (nfa address -- , copy name to address as string)

Accepts a name field address and an address, and copies the name field to the address, then fixes it like a string. Allows the name to be printed as a string in graphics applications.

Related Words: NFA->$

NFA->$ (nfa -- $string , convert NFA to useable string)

Copies name to PAD and strips bad bits from dictionary version.

PFA->NFA (pfa -- nfa)

Converts PFA to NFA. See CFA->LFA.

****

Note (Pick us a winner): !!! Note on following variants of PICK: !!!

PICK in Forth83 is different from PICK in Forth79. HForth and JForth PICK's conform to Forth83.

****

PICK79 (vn vn-1 … v1 x -- vn vn-1 … v1 vx ,)
(returns the x'th item on the stack, starting its count at 1)

Takes x off the stack, finds the x'th item on the stack and duplicates it.

```
33 45 62 80 2 PICK79 .   \ prints 62
```

Related Words: PICK83 is closely related.

Macintosh/Amiga: PICK83 is the same as PICK in Forth 83.

PICK83 (vn vn-1 … v1 x -- vn vn-1 … v1 vx ,)
(returns the x'th item on the stack, starting its count at 0)

Takes one number off the stack, goes that deep into the stack and copies the value there onto the top of the stack. In HMSL (HForth and JForth, **PICK is the same as PICK83**).

NOTE: The difference between PICK83 and PICK79 is that PICK83 starts counting at zero, and PICK79 starts counting at one!

```
33 45 62 80 2 PICK83 .  \ prints 45
```

Related Word: PICK79 is closely related

Macintosh/Amiga: PICK in Forth 83 is the same as PICK83.

SAFE.EMIT (char -- , emits if safe or emits a '.')

Accepts a character, decides whether it will mess up the screen or printer if you try to emit it. EMITs it if it's safe, otherwise emits ".".

SERVICE.TASKS (-- , execute low level tasks)

This is one of the basic words for vectored execution in HMSL. It is used within a looped routine when you want some other lower level task to be performed at intervals during your loop.

v *Warning:* This example is provided as an interesting systems application. You should never alter TASKS-CFA yourself, or you may do something disastrous to HMSL. TASKS-CFA is used by the main words which run HMSL, and should be left alone. We provide this documentation because it provides a valuable technique which users may adapt to their own needs.

Example: You want to draw a graph while playing some music. First, you define MAKE.SOUNDS to produce the music. (Note that this is a "general" example—don't try to type it in and run it.)

```
:  DRAW  ( N -- )
   DO DRAW.SOMETHING
      SERVICE.TASKS DRAW.MORE
   LOOP
;

:  INIT.SERVICE.TASKS
```

```
        'C MAKE.SOUNDS TASKS-CFA!
    ;
```

If you just want to draw, then you can leave SERVICE.TASKS set to NOOP, and DRAW will function as you expect. If you want to make music at the same time, then first you should type: INIT.SERVICE.TASKS, then call DRAW. What you will have done, in effect, is to bypass Forth's requirement of defining a word before you use it inside another routine.

HMSL uses SERVICE-TASKS and TASKS-CFA to set up the basic structure of high level routines before the details of the internals of those routines are fully defined. We "stuff" TASKS-CFA late in the code, but we have routines which call SERVICE.TASKS, which is defined much earlier on as:

```
    : SERVICE.TASKS TASKS-CFA @ EXECUTE ;
```

Again, **do not** change TASKS-CFA yourself if you are using HMSL unless you ABSOLUTELY know what you are doing, and are ready for some serious repercussions. Changing this word is tantamount to changing the fundamental HMSL scheduling process.

SERVICE.TASKS/16 (-- , service tasks at 1/16th rate)

Like SERVICE.TASKS, but only does "service-tasking" every 16 counts. See explanation of SERVICE.TASKS above.

SET.BITS (flag mask value -- value , sets or resets bits)

The flag should be 1 or 0. The mask and value should be binary numbers. The digits in the mask determine which bits of the value will be affected. The flag determines whether these bits will be made into 1's or 0's.

Example 1:

```
    BINARY 1  10011010  00000101  SET.BITS
```

results in: 10011111

Example 2:

```
    BINARY 0  00000010  00001111  SET.BITS
```

results in: 00001101

Macintosh/Amiga: JForth word, defined in HMSL for Macintosh.

STACK.CHECK (-- checks to make sure the stack is OK)

Issues error message -- " Change in stack depth!" -- if stack depth has changed. Looks at marker set by STACK.MARK, so STACK.MARK should be used before doing a tricky word, then use STACK.CHECK. This word is handy for checking whether a loop is accidentally leaving things on, or eating from the stack. If you do 60,000 loops and accidentally leave a number on the stack every time through then you can get in real trouble.

NOTE that STACK.CHECK and STACK.MARK are both used by the HMSL main loop to check that user code hasn't corrupted the stack. For that reason, you should avoid using STACK.MARK in any words that are embedded in the HMSL loop. Feel free to use them in words that operate independently of the HMSL tasking environment, however.

Related Word: STACK.MARK

STACK.MARK (-- , record depth of stack)

Stores depth of stack in a variable called STACK-HOLD. Used in conjunction with STACK.CHECK to see if the stack has been corrupted. See above description of STACK.CHECK.

TAB (-- , emit horizontal tab character)

Moves cursor up to 9 spaces over. Like a typewriter tab.

TEXT>STRING (address count -- $string)

Converts some characters to a Forth string by copying them to the pad with a count byte.

TOLOWER (char -- lowercase-char, converts to lowercase)

Accepts any character, uppercase or lowercase, returns that character as lowercase

TOUPPER (char -- uppercase-char , converts to uppercase)

Accepts any character, uppercase or lowercase, returns that character as uppercase.

V: (--)

Synonymous with VARIABLE. Saves typing. Creates a named variable with one cell storage allocation. Used outside colon definitions, before using the variable. This was defined because in some old Forths, VARIABLE expects an initial value on the stack which is different than Forth 83 which doesn't. V: is thus more generic.

Example: V: NEXT-NOTE (creates a variable named NEXT-NOTE)