HForth Glossary

Introduction

This glossary defines the low level Forth words that are part of HForth. Music related words will be defined in the HMSL Manual. Some words, specific file I/O related words for example, will be defined in this manual in the chapter on File I/O. In many cases, however, there will be a brief reference definition and a reference to another section.

Many of these words are standard in most Forths. If you cannot remember the exact name of a word, use **WORDS.LIKE** to find all words containing a substring.

Stack Diagrams

Most words are defined in terms of their stack diagrams. A stack diagram tells you what parameters are passed on the stack, and what is returned. Consider the stack diagram for @ which is used to get a value from memory.

```
@ ( addr -- n , fetch value from address )
```

Notice that the stack diagram is enclosed in parentheses so it is a legal Forth comment. If you look in the HMSL source code, you should see stack diagrams for the definitions of every word. Put stack diagrams on every word you define.

The word ADDR is to the left of the "--" which means that it is passed to @. Everything after the "--" is returned from the word Thus @ takes an address ADDR and returns a value N. If there is more than one parameter, the rightmost will be on the top of the stack. Thus:

```
- ( a b -- a-b , subtract b from a )
```

To use minus you would enter:

```
10 3 - . ( prints 7 )
```

In this case b is 3. It is on top of the stack and is the rightmost parameter in the stack diagram.

We try to use consistent names for parameters in this glossary. They are:

```
$string  -  address of count byte of a text string
<name>   -  name follows the word, prefix notation
<filename> -   filename that follows the word
addr     -  address
d     -  double precision number
n     -  number
n -r- -  return stack, see >R
u     -  signed number
```

HMSL Source Code Directories

Since the source code for most of HMSL is provided, you can usually find the stack diagram of a word. There is a tool called **FILE?** that will tell you what file the word is defined in.

```
FILE?  HMSL.START
```

Note that words defined in the assembly language kernel will not be available for viewing.

Most of the files can be found in one of the following directories:

H: HMSL_Source:Source - HMSL code, morphs, etc.

HH: HMSL_Source:HostDep - Mac specific

HO: HMSL_Source:ODE - ODE support

! " # $ % & ` ( ) * + ' - . / 0 - 9 : ; < = > ? @ A Z [/] ^ _ a z { | } ~

HForth Glossary

! (n addr -- , store n at addr)

Store the 32 bit value N into memory at ADDR. ADDR must be even.

```
VARIABLE VAR1
765 VAR1 !
VAR1 @ .  \ prints 765
```

Related Words: @ W! C!

“ (<string”> -- addr)

Declare a string. The address points to a count byte which is followed by the text.

```
“  Hello”  COUNT  TYPE
```

Warning: if " is used outside a colon definition, the string is stored at PAD which is a temporary location. If you create two strings the second string will overwrite the first one.

```
" Hello" $TYPE
" Hello" " frog" $TYPE $TYPE  ( prints frog twice !! )
: T " Hello" " frog" $TYPE $TYPE ;
T ( prints Hello frog )
```

 # (d1 -- d2)

Convert one more digits of a number being converted to a string. Divides D1 by BASE, converts the remainder to an ASCII character and calls HOLD to add it to the string. Here is an example that prints a number with leading zeros.

```
: .W  ( n -- , print with leading zeros )
   s->d <#  # # # #S #>
TYPE SPACE
;
HEX 0020 000F OR .W  ( prints 002F )
```

#> (d -- addr count)

Terminate a numeric conversion. Return numeric string.

#S (d1 -- 0.0)

Completes the conversion of a double number to text. See #.

#TIB (-- var-addr , number of characters in TIB)

$ (<number> -- , for hex number entry)

Treats the next string as a hexadecimal number. For example:

```
$ 1FAE  DUP  .  .HEX
$ 10 .  10 .
```

$= ($1 $2 -- flag)

True if strings identical. Case sensitive.

Related words: TEXT= COMPARE MATCH?

$APPEND (addr count $string -- , append to string)

Make sure there is room at the end of the string for the added text.

$ARRAY (#strings #chars_max --)

Make an array for strings.

```
10 40 $ARRAY $STRS
```

```
" Hello" 5 $STRS $MOVE
5 $STRS COUNT TYPE  ( prints: Hello )
```

$FOPEN ($filename -- fileid , open file)

See the chapter on File I/O.

$LOGTO ($filename -- , send output to the file)

See the section of LOGTO in the Forth Tools chapter of the HMSL Manual.

$MATCH ($string1 $string2 -- flag)

Return true if strings match. Case INsensitive.

$MOVE ($string dest-addr -- , copy string to destination)

$ROM (<name> -- , when created)

(index -- $string , when used)

Creates a read-only string table. For example:

```
$ROM  MESSAGES  ,"  Hello" ,"  Abandon ship!"  ," Who am I?"
2 MESSAGES $TYPE
```

Related words: TEXTROM ," $ARRAY

$TYPE ($string -- , print string)

' (<word> - cfa)

Look up word in dictionary. Return address of executable code. Same as 'C

```
: HI ." Hello" ;
' HI EXECUTE  ( prints hello )
```

((-- , start a comment)

Everything between parentheses will be considered a comment.

*** (a b -- a*b , 32 bit multiply)**

***/ (a b c -- a*b/c , uses 64 bit intermediate product)**

This is useful when the intermediate result of the multiply will exceed the 32 bit limit.

***/MOD (a b c -- remainder a*b/c)**

+ (a b -- a+b , 32 bit add)

+! (n addr -- , add N to value at ADDR)

```
VARIABLE  VAR1
1  VAR1  +!   ( add one to contents of VAR1 )
```

, (n --) "comma"

Compile N into dictionary at current position. This is handy for making tables. For example:

```
CREATE DATA  23 , 79 , 172 , 944 ,
DATA  3 CELLS + @ .  ( prints 944 )
```

," (string" -- , compile string into dictionary)

Can be used to create permanent strings or string tables. For example:

HERE ," Hello!" $TYPE

-2SORT (a b -- largest smallest , reverse sort, see 2SORT)

-> (n <name> --) "to"

Set a *local variable* or a *value* to N.

```
75 VALUE  MYVAL
23 -> MYVAL
```

```
MYVAL .  ( prints 23 )
```
Related Words: { VALUE

-ROT (a b c -- c a b)

Equivalent to doing ROT twice but faster.

. (n -- , print number N) "dot"

Prints N in the current numeric base.

```
100 5 / .  ( prints 20 )
```
Related words: .R N>TEXT #

." (<string"> -- , print string)

Used for printing messages.

```
." Hello Fred!" CR
```

.. (addr <member> -- addr-member)

Generate the address of a structure member (not an HMSL structure, but a Macintosh structure).

See chapter on Mac Internals.

..! (n addr <member> --)

Store N into the named member of the structure whose address is given.

```
200  MY-THING ..!  TH_WIDTH
```

See chapter on Mac Internals.

..@ (addr <member> -- n)

Fetch N from the named member of the structure whose address is given.

```
MY-THING ..@  TH_WIDTH . ( prints 200 )
```

.ELSE (-- , see .IF)

.HEX (n -- , print N using base 16)

.IF (flag -- , for conditional compilation)

If flag is true, compile until a .THEN or .ELSE is encountered. For example:

```
HOST=MAC   .IF
: OPEN.WINDOW  ( mac specific code ) ;
.ELSE
: OPEN.WINDOW  ( other host code ) ;
.THEN
```

.R (n width -- , print right justified)

Print N in a field WIDTH characters wide. Useful for generating formatted tables.

.S (... -- ...)

Print contents of stack nondestructively.

.THEN (-- , see .IF)

/ (a b -- a/b , 32 bit divide)

Please note that with integer math the answer will be truncated so the order of operations is important. For example:

```
1 2 / . ( will print 0 )
2 3 /  10 * . ( will print 0 )
10  2 * 3 / . ( will print 6 )
```

/MOD (a b -- remainder a/b)

0< (n -- flag , true if less than 0)

0= (n -- flag , true if equal to 0)

0> (n -- flag , true if greater than 0)

0SP (x y z w q etc -- , clears stack)

Use this instead of SP! (which is defined differently than in FIG Forth).

1+ (n -- n+1 , add 1 , fast)

1- (n -- n-1 , subtract 1 , fast)

2* (n -- n*2 , multiply by 2 , fast)

2+ (n -- n+2 , add 2 , fast)

2- (n -- n-2 , subtract 2 , fast)

2/ (n -- n/2 , divide by 2 , fast)

2DROP (a b -- , drop 2 numbers)

2DUP (a b -- a b a b , duplicate pair of numbers)

2OVER (a b c d -- a b c d a b)

2SORT (big little | little big -- little big)

Sorts the two top items on the stack, leaving the largest on top. Opposite of -2SORT .

2SWAP (a b c d -- c d a b , swap 2 pairs of numbers)

4* (n -- n*4)

4+ (n -- n+4)

4- (n -- n-4)

4/ (n -- n/4)

Example: 5 4/ . Prints "1". No remainder is put on the stack.

: (<name> -- , begin defining a Forth word)

Creates entry in dictionary. Set SMUDGE bit so it can't be found Sets STATE to TRUE. Terminated with ;

Related words: LATEST UNSMUDGE WORDS ; CREATE

:STRUCT (-- , see chapter on Mac Internals)

; (-- , terminate a colon definition)

Set STATE to FALSE. Unsmudge last word.

< (a b -- flag , true if A less than B)

= (a b -- flag , true if A equal to B)

> (a b -- flag , true if A greater than B)

>BODY (cfa -- pfa)

Convert CFA to "Parameter Field Address" which is the data portion of a CREATE DOES> word.

>IN (-- var-addr , index of current char in TIB)

>NAME (cfa -- nfa)

Convert CFA to "name field address" by scanning dictionary for nearest name. This can be quite slow because HForth uses separate headers. For example:

```
'  NEGATE  >NAME  ID.
```

Related words: NAME> >BODY >LINK

>NEWLINE (-- , move cursor to new line if needed)

Related words: CR OUT CR?

>R (n -- , -r- n)

Move N to return stack. The return stack is used by the 68000 subroutine calling mechanism so you must remove this number using R> or RDROP before returning. See also: R@

?COMP (-- , verify compilation mode)

Pronounced "question compile". If the system is not in compile mode as defined by STATE containing a non-zero value, then it issues an error message, dumps and aborts.

?DUP (0 -- 0 | n -- n n , DUP if non-zero)

?PAUSE (-- , pause if key hit)

This will pause if a key was hit. It is handy for putting in a loop that is printing lots of text. HMSL uses it when PRINT:ing array contents.

Hit SPACE to continue after pausing. Any other key will abort.

?STOP (-- flag , like ?PAUSE but doesn't abort)

```
: EXAMPLE   BEGIN  ."  Hello" CR ?STOP  UNTIL ;
```

?TERMINAL (-- flag , true if keyboard hit)

KEY can then be called to get the character.

@ (addr -- n , 32 bit fetch from memory)

The address must be even. For example:

```
variable VAR-1  876  VAR-1  !
VAR-1  @  .  ( prints 876 )
```

?QUIT (-- flag , just ask if we want to quit with 'Q')

ABORT (-- , terminate execution)

Return to interactive Forth interpreter. Clear stack. Deferred word.

ABORT" (flag <string"> --)

ABORT and display string if flag is true.

ABS (n -- |n| , absolute value of N)

Make N positive if negative.

ALIGN (-- , Makes DP even)

Adds one to the DP (dictionary pointer) if necessary to make it even. You might need this word if you were allocating bytes one at a time, as when using C, or ALLOT. The 68000 can generate address errors if the DP is left odd.

CREATE FOOTAB 55 C, 66 C, 77 C, ALIGN

Related Words: C, ALLOT HERE EVEN-UP

AGAIN (--)

Used with BEGIN to loop forever. RARELY USED!

Related words: EXIT UNTIL

ALLOT (n -- , alloacte N bytes in dictionary)

Allocate memory by advancing dictionary pointer. For example:

```
    CREATE TABLE  100 ALLOT    ( make 100 byte table )
```

See also: ALIGN DP HERE ,

AND (a b -- a&b , perform 32 bit logical AND)

If the same bit is one in A AND B then it will be one in the result, otherwise it will be zero. This is usually used for masking part of a word.

```
    $ ABCD  $ FF AND  .HEX   ( mask off high bits, print CD  )
```

Related words: OR XOR .HEX BINARY

ANEW (<name> --)

FORGETs a word if already defined, then defines it. This is often used at the beginning of a file that is under development. The filename is typically appended to the prefix TASK- . A simple example would be

```
    ANEW TASK-MY_FILE  ( where MY_FILE is the name of the file )
```

followed by the code in the file. ANEW should be the first line of executable code in the file, and it is a good stylistic convention to use the name of the file itself, preceded by TASK- for the "fence" word. ANEW is a nicer version of the conventional Forth phrase:

```
    :  XXX  ;  FORGET XXX
```

placed at the beginning of the file.

ARRAY (#cells <name> -- , create named array)

Create an array in the dictionary #cells long. Here is how ARRAY is defined:

```
: ARRAY  ( #cells <name> -- )
   CREATE  cells allot
   DOES>  ( index addr -- addr' )
      swap cells +
;
```

Here is how it is used:

```
100 ARRAY  MYDATA
\  MYDATA  ( index-cell -- addr-cell )
6152 22 MYDATA !
22 MYDATA @ . ( print 6152 )
```

ASCII (<char> -- n)

Every character has a number associated with it that is used by the computer. If you are doing letter comparisons or other text operations you will need this ASCII number.

```
ASCII W .
KEY ASCII A = .
```

ASHIFT (number shift-count -- shifted-number)

Arithmetically shifts the number to the left or to the right, by the number of bits specified in shift-count.

Shifts toward the most significant bit and shifts zeros into the least significant bit if shift-count is positive. Shifts toward the least significant bit and shifts ones into the most significant bit if shift-count is negative. Number is a two's complement number. This differs from SHIFT in that it preserves the sign for negative shift count.

Provides a quick multiply or divide by a power of 2.

```
 3 2 ASHIFT .    ( prints 12 )
-100 -2 ASHIFT .     ( prints -25 , different from SHIFT )
```

Related words: all / and * words and SHIFT

AUTO.INIT (--)

When HForth first starts up, it looks in the dictionary for a word called AUTO.INIT and executes it. This word should first call AUTO.INIT and can then perform some user defined initialization. Since

it calls AUTO.INIT, a chain of initializations is formed. (Note that AUTO.INIT is not called in TURNKEYed programs since there is no dictionary to search.) For example:

```
... FOO related code...
\ Initialize all previous systems then FOO
: AUTO.INIT  auto.init  foo.init ;
... MOO related code ...
\ Iinitialize all, including FOO, then MOO
: AUTO.INIT auto.init moo.init ;
```

AUTO.TERM (--)

Just like AUTO.INIT except this is called when you QUIT HMSL. You should terminate your system first, THEN call AUTO.TERM.

```
: AUTO.TERM  moo.term  auto.term ;
```

Related Words: IF.FORGOTTEN

ASSIGN (<new_name:> <full_name:> -- assign logical volume name)

This name can be used with INCLUDE and INCLUDE? and FOPEN.

As an example, suppose you wanted to compile code from a directory called HARD_DISK:HMSL_WORK:JOE:PIECES:WATERFALL , you could enter:

```
ASSIGN  WF:  HARD_DISK:HMSL_WORK:JOE:PIECES:WATERFALL
```

Then you could put these commands in a file:

```
INCLUDE?  TASK-WATER_1 WF:WATER_1
INCLUDE?  TASK-WATER_2 WF:WATER_2
```

You could later move the code to other disks. Then by simply updating your ASSIGN you can use the same loader. This word is what allows HMSL to be compiled either from a hard disk or from a floppy.

One restriction, the names must not have any spaces in them!

An assignment can be cleared by leaving the full_name blank, eg.

```
ASSIGN WF:
```

ASSIGNS? (-- , print assigns)

Prints a table of the current names defined by ASSIGN .

BASE (-- addr)

A variable containing the current numeric base used for input and output conversion. DECIMAL sets BASE to ten. HEX sets it to 16. BINARY sets it to 2.

```
5  BASE  !  ( set to base 5 )
4  1  +  .  ( should be 10 )
HEX   100  DECIMAL  .  ( should print 256 )
```

Here's a puzzle! Predict the output of the following, then try it out. You may be surprised!

```
DECIMAL BASE @ .
HEX BASE @ .
HEX BASE @ DECIMAL .
```

BEGIN (--)

Begin a program loop. This must be used with UNTIL, or WHILE REPEAT, or AGAIN. See those words for examples.

BINARY (--)

Set numeric BASE to two.

```
BINARY  1001  11 + .  ( prints 1100 )
```

BL (-- 32)

Constant equal to the ASCII value of a BLANK or SPACE.

BODY> (data-addr -- cfa)

Converts the address of the data area of a Forth word created using CREATE to its code field address.

```
CREATE  FOO 723 ,
FOO  BODY>  >NAME  ID.  ( print FOO )
```

Related Words: >NAME >BODY ' >LINK

BYE (--)

Exit HMSL. This will call the AUTO.TERM chain unless TURNKEYED.

C! (byte addr --)

Store byte at given address in memory. Only the lower 8 bits are used.

C@ (addr -- byte)

Fetch a byte from the given address in memory. The upper 24 bits will be set to zero.

```
" Hello" 1+ C@ EMIT  ( print 'H' )
```

C, (byte -- , compiles into dictionary)

Compiles one byte into the dictionary at the current dictionary location. As is the case in all 32- or 16-bit Forths, you must always use an even number of calls to C, or use ALIGN. DP, the dictionary pointer, must always be an even number.

Example: When filling a byte table, you could enter:

```
CREATE MY-BYTES  5 C, 8 C, 13 C, 21 C,
```

Related Word: ALIGN

CASE (n -- n)

CASE is a conditional construct for selecting one of several possible actions based on a value N. Put default or illegal value processing after last OF statement. After an OF or ?OF or RANGEOF is satisfied, execution will skip to ENDCASE.

```
:  HANDLE.N  ( n -- )
   CASE
      1 OF  ." Equals 1 !" CR ENDOF
      2 OF  ." N=2" cr ENDOF
      5 10 RANGEOF  ." In valid range!" ENDOF
      dup 10 > ?OF ." Greater then 10 ENDOF
      ." Illegal value = " dup . CR
   ENDCASE
;
```

CELL (-- 4)

Constant equal to the width of the data stack in bytes. In 32 bit Forths, CELL = 4 (HForth and JForth). In 16 bit Forths, CELL = 2.

CELL* (number -- number*cell)

Multiply by value of CELL (4 in 32-bit Forths, 2 in 16 bit Forths).

Used when indexing into an array by the specified number of cells. Multiplies the value on the stack by CELL.

```
5 CELL* .  prints 20
```

Related Words: CELL+, CELL-

CELL+ (number -- number+cell)

Adds CELL to top of stack -- i.e. increments by one cell.

CELL- (number -- number-cell)

Subtracts CELL from the top of the stack -- i.e. decrements by one cell.

CD (<volume-name> -- , set default folder)

Equivalent to UNIX cd command to "Change Directory".

CHOOSE (n -- random)

Choose a random number between 0 and N-1 inclusive. CHOOSE generates a pseudo-random number sequence using a linear congruential method. It repeats after 65536 calls. CHOOSE keeps its seed in a variable called RAND-SEED which can be preset to give the same sequence every time. The maximum value for N is 65536.

```
: ROLL.DIE  ( -- dice# )   6 CHOOSE 1+ ;
```

Related Words: WCHOOSE

CLS (--)

Clear the text screen.

CMOVE (source-addr dest-addr #bytes --)

Move #BYTES of data from the SOURCE address to the DESTINATION address, starting with the lowest address. WARNING: If the source and destination region overlap and dest>source then you will overwrite part of the data as you write it. Use CMOVE> if this is not desired. MOVE is a smart version of CMOVE that chooses the right direction.

CMOVE> (source-addr dest-addr #bytes --)

Move #BYTES of data from the SOURCE address to the DESTINATION address, starting with the highest address. WARNING: If the source and destination region overlap and dest<source then you will overwrite part of the data as you write it. Use CMOVE if this is not desired.

COMPARE (addr1 addr2 #bytes -- result)

Compare two strings of bytes and return a result that indicates their alphabetical relationship. This is useful when sorting items into alphabetical order. CASE SENSITIVE!

```
if S1 = S2, then r = 0
if S1 > S2, then r = 1
if S1 < S2, then r = -1
: TEST    " peak"  count drop " frog"  count compare . ;
TEST   ( print 1 )
```

Related words: MATCH? TEXT=? SCAN SKIP $=

COMPILE (<name> --)

Compile a call to the named Forth word. This is used to write a word that compiles other Forth words. This word is IMMEDIATE and can only be used at compile time. It is mostly used for internal system level programming. Here is an example of a word that redefines LOOP so that it will abort if you hit a key.

```
: CHECK.ABORT  ( -- , abort if key hit )
   ?terminal abort"  Key hit while LOOPing!"
;
: LOOP   ( -- , redefine to avoid endless LOOPS )
   ." Compiling Safe LOOP" cr
   COMPILE CHECK.ABORT
   COMPILE LOOP
;  IMMEDIATE
:  TEST  1000000 0 DO ." Hello" cr LOOP ;
TEST  ( Normally a bad idea. Just hit a key. )
```

Related Words: [COMPILE] STATE CREATE

CONSTANT (n <name> --)

Define a Forth word that will return N when called. It is highly recommended that you use constants for program values to make code more readable and easier to change.

```
20 CONSTANT NUM_THINGS
NUM_THINGS .  ( print 20 )
```

Related words: VALUE

COUNT ($string -- addr #bytes)

Forth text strings are stored as a count byte followed by the text. This convert the address of the count byte to the address of the first character and the number of bytes.

```
" Hello" 10 DUMP   ( see internal structure )
" Hello" COUNT  .S  ( prints:  address-of-H  5  )
```

CR (--)

Output a Carriage Return character to the output device. A carriage return is an ASCII $ 0D. This causes the following text to start printing on the next line.

```
."  line 1" CR ." line 2"
```

Related Words: >NEWLINE CLS EMIT EOL CR?

CR? (-- , print carriage return if needed)

Only does a CR if the cursor is near the end of a line.

CREATE (<name> --)

Creates a new Forth word in the dictionary. By default, this word will simply return the address in the dictionary immediately following. If you place data there you can use this word to make data structures. It is the building block for ARRAY and much of ODE.

```
CREATE FOO  234 , 999 ,
FOO @ .  ( print 234 )
FOO cell+ @ .  ( print 999 )
```

See the chapter on Forth internals and dictionary structure. See also the definition of DOES> which allows you to make smart data structures.

Related Words: DOES> VARIABLE ARRAY >BODY 'P

CURRENT (-- address , of last word in dictionary)

Newly defined words will be linked to this word. The new word will then become the CURRENT word.

```
: FOO ;
CURRENT @ ID.  ( print FOO )
LATEST ID.   ( print FOO too )
```

Related Words: LATEST CONTEXT HERE