HAL/S Interpreter Guide

By Thomas Curry

[Image result for nasa worm](https://www.google.co.uk/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwim54LsiObTAhXLJ8AKHWuGAO0QjRwIBw&url=http://kottke.org/15/08/nasas-logo-the-worm-vs-the-meatball&psig=AFQjCNFNgIhgee4y6eVx13DkJ1xsr--VdA&ust=1494531132890056)(Unafilliated with )

## Interpreter rules (quibbles)

* Everything MUST be in UPPER CASE. This appears to be standard for all HAL /S, which does not (in any of the examples I found) have lower case.
* Every line of code should be set out as \*prefix\* \*space\* \*code\* so for example:

M WRITE(1) OUTPUT;

* Valid line prefixes are C (comment), E (exponent), S (subscript), M (main (between E and S)) and D(?).
* EMS form is not valid anymore (yet) due to the new arithmetic system.
* \*\* = ^ (This was originally HAL/S syntax.) (As in \*\* equals power)
* All lines should end in a semicolon (;). This is standard HAL /S and while some lines of code are not checked for the existence of one, some are.
* Equals (=) MUST be surrounded by spaces I.E OPERAND1 = OPERAND2 not operand1=operand2.
* Expressions MUST NOT have spaces, so A\*T is valid, A \* T is illegal.
* Using both of these rules, it follows that equations therefore must take the form VARIABLENAME = X\*Y. This also applies to logical operations such as IF x > 9 THEN DO;
* Values in standard form can be expressed as 4.5E23, however negative standard form is invalid (4.5E-23) so a variable will have to be defined as (for example) 10\*\*-23.
* Fractional powers are invalid (I.E X\*\*1/2) but decimal powers are (I.E X\*\*0.5)
* As of the time of writing, there can be no indentation, for example in if statements.
* At time of writing, there are no simple (one line) if statements, and all If statements have to have an else.
* ~~If statements cannot appear within loops, unless there are no other statements in the loop AFTER the if statement (due to conflicts with the END; statement).~~ Fixed thanks to stacks!
* ~~If statements have to have an else.~~ Not anymore!
* There is also no array declaration (but it will be added(hopefully!)).
* Trigonometric functions have support for both single number use (SIN(1.57)) and equation use (SIN(3.14/2)). Trigonometric functions take radian inputs. ~~NOTE – Trigonometric functions are currently not implemented (anymore), as they are built in functions, and functions have yet to be overhauled and implemented into the new arithmetic system. (but they are getting there!).~~ Built in functions are back!
* When using FUNCTION, there should be no spaces when defining parameters (e.g. FUNCTION(VALUE1,VALUE2)).
* I have also added a new keyword – RUN. This allows a program to run another (HAL/S) program.
* Pressing F10 automatically refactors the code (think auto-indentation).
* ~~And finally the big downside – You can (for now) only use two operands in arithmetic expressions.~~ Now fixed with the new arithmetic system (yay!).

## Valid operators

The valid operators are currently:

|  |  |  |
| --- | --- | --- |
| Operator | Operation | Example |
| + | Addition | 1+1 = 2 |
| - | Subtraction | 3-1 = 2 |
| / | Division | 6/2 = 3 |
| \* | Multiplication[[1]](#footnote-1) | 7\*3 = 21 |
| \*\* | Exponentiation (Powers) | 2\*\*8 = 256 |

## Comparators

The following operators can be used for comparison:

|  |  |
| --- | --- |
| = | Equals |
| > | Greater than |
| < | Less Than |
| >= | Greater than or equal to |
| <= | Less than or equal to |

## Appendix 1

All valid Keywords (so far):

* SIMPLE (does nothing yet)
* DECLARE
* READ
* WRITE
* IF
* ELSE
* END
* DO FOR
* DO WHILE
* *FUNCTIONNAME:*
* FUNCTION
* CLOSE
* *X = ...*
* RUN
* WAIT *Time in miliseconds* E.G WAIT 1000 waits for 1 second
* LOAD (Again, I added this. Loads vars from MMU into memory)

To be added:

* REPLACE (maybe – no real need to implement it)
* SIMPLE IF
* UNTIL
* EXIT

## Appendix II -Syntaxes

### Conditional Execution:

IF *(LOGICAL STATEMENT)* THEN DO;

‘ ‘ ‘

END;

### Iteration:

DO FOR I = 1 TO 10;

‘ ‘ ‘

END;

DO WHILE I < 10;

‘ ‘ ‘

END;

### Main Program code:

(note – equivalent of vb.net Sub Main())

SIMPLE: *NAMEOFPROGRAM;*

‘ ‘ ‘

CLOSE SIMPLE;

### Function / subroutine

FUNCTIONNAME: FUNCTION(PARAMETER1,PARAMETER2,*ETC*) *TYPE*;

CLOSE *FUNCTIONNAME;*

### Input

READ(*CHANNELNUMBER)* *VARIABLETOASSIGNTO*

### Output

WRITE(*CHANNELNUMBER)* ‘*String’*

Or

WRITE(CHANNELNUMBER) *ARITHMETICEQUATION* (or variable name, or whatever)

## Appendix III – MMU

The MMU (mass memory unit) was a large store for data critical to the shuttle PASS. It could be accessed by all 5 computers. There were two installed on the orbiter, each could store 8 million 16 bit words (Around 16Mb). In this emulation it is represented by a text file. Channel 9 is the direct channel to the MMU (Chosen arbitrarily by me, this isn’t a real orbiter). The command “LOAD” loads all the variables in the MMU, available for use by the current program. This effectively loads the data into the emulated “computer” ram.

1. HAL/S had support for a space, “.” or in some instances no gap to represent multiplication. How they separated the variables is beyond me! Due to the way the interpreter works, using a space is impossible (as is sadly, full stops due to the conflicts they create with decimal points). [↑](#footnote-ref-1)