HUBBARD
Software

CLI
Advanced Command Line Interpreter

A powerful alternative to the standard system 'shell', the Hubbard Software CLI is positioned between limited shells and scripting languages. It contains a robust set of native functions, can be run interactively like a shell, or using command 'macros' (batch files). Plus, it is portable across Operating Systems, as it is written completely in Java®.

Available in several packages from a free version up to an Enterprise version which allows full user customization. In the upper versions, you can even add your own commands and localize all the text and formatting for your own requirements. If you don't like our command names, just rename them!

Power Features:

- Robust command set – No need for many external tools.
- Macros – Write your own batch scripts with powerful argument notation. No more 'shifting'!
- Named Variables – String, Integer, and Floating Point. Use the default or name as many as you need!
- Full User Environment Stack – Variables, Working Directory, Prompt, Output File, and many other settings live on a stack you can push and pop from.
- Minimal Uniqueness on all Commands and Switches – Tired of typing?
- Pseudomacros – A large set of 'functions' that can be used anywhere in a command, and nested. Includes If/Then/Else control flow structures, String handling, Variables, Math, File System info, Date/Time handling, and more.
- Structured Error Handling – Need to ignore an error on occasion? Or even treat a warning more seriously? No problem. Available per command or for whole session.
- Output Redirection – Available for whole session, or just per command.
- Full Localization and Customization available.

More Features:

- Portable! Runs on any OS that supports Java.
- Powerful command notation shortcuts: Parentheses and Angle Brackets:

  Parentheses repeat the command for each token inside them:

) delete Temp_(PO INV).dat

  Repeats the delete command for TEMP_PO.dat and TEMP_INV.dat .

  Angle brackets concatenates strings together:

) write Test_<a b c>
  Test_a Test_b Test_c

  Or even:

) write Test_<a b c>.<tmp bak>
  Test_a.tmp Test_a.bak Test_b.tmp Test_b.bak Test_c.tmp Test_c.bak

  And you can combine both!

- Searchlist – Much like a 'path' but specific for use by the CLI. Lives on the stack.
- Math Pseudomacros – Like [!iadd 2 3], or even [!log10 10000].
- String manipulation – Like [!string/name=mystring/start=3] for a substring. Or find the index of a substring like this where we set the default string variable to 'abcdef' and then look for the substring 'def' in it.

) string abcdef
) write Position of 'def' is: [!string/index def]
  Position of 'def' is: 3

- Execution control – If/Then/Else tests for all data types. For example:

) write A and B are [!equal A B]Equal[!else]Not Equal[!end]
  A and B are Not Equal

Packages

	Free	Basic	Commercial	Enterprise
File Handling	☺	☺	☺	☺
Searchlist	☺	☺	☺	☺
Pseudomacros	☺	☺	☺	☺
Macros		☺	☺	☺
String Functions		☺	☺	☺
Format Control		☺	☺	☺
Run Programs		☺	☺	☺
Advanced Commands		☺	☺	☺
Custom Configurations			☺	☺
Localization				☺
Write your own code				☺
Number of Seats[1]	**Unlimited**	**1**	**50**	**Unlimited[2]**

Notes:

[1] - A "Seat" is either a workstation (or virtual workstation) used by a human, or a server used for things like File Services, Database Servers, Web Servers, and the like. It can have any number of 'Cores'. On a "Virtual Machine" server, each Operating System instance counts as a Seat.

[2] -"Enterprise" package includes 8 hours of development assistance.

Examples:

Use named variables, Pseudomacros, and conditional expansion.

) string/name=str1 AAA
) string/name=str2 BBB
) write String 1 and String 2 are [!equal [!string/name=str1] [!string/name=str2]]Equal[!else]Not Equal[!end]
String 1 and String 2 are Not Equal

Note the use of the [!equal], [!else], [!end], and [!string] 'pseudomacros' that allow inline processing within a command string, nested as needed.

Repeat a command or macro for multiple arguments by putting the args in parentheses.

Here, a batch script (macro) named invoice_run.cli is being run twice, once with each file as an argument.

) invoice_run.cli (jan_invoice.dat feb_invoice.dat)

Run multiple commands on the same arguments.

) (filestatus type) invoice.dat

Or using minimal uniqueness:

) (fi ty) invoice.dat

Use shortcuts to make multiple arguments from concatenated strings by putting text in angle brackets.

) write Test.<dat txt>
Test.dat Test.txt
) write Test<1 2>.<dat txt>
Test1.dat Test1.txt Test2.dat Test2.txt

And here we show how the above example of an invoice run is done by combining the parens and the angle brackets:

) invoice_run.cli (<jan feb>_invoice.dat)

Run a backup, logging the details to a file.

) push
) cd c:\users\martin\Documents
) listfile backup_[!date/format=yyyyMMdd]_[!time/format=HHmmss].log
) move/copy/recent/v/exclude=Dev "F:\\Laptop Docs" #
) pop
We pushed a level of the environment, changed directory, set the output (listfile) to a log file with date/time stamp, then moved any files that changed, where '#' means the current directory and everything under it, and we exclude the Dev directory. Then popped the environment, restoring us to the previous level working directory, and also restoring the previous listfile (possibly your console). Note the use of the [!date] and [!time] 'pseudomacros'.

Or now the same thing using minimal uniqueness:
) pu
) cd c:\users\martin\Documents
) li backup_[!da/f=yyyyMMdd]_[!ti/f=HHmmss].log
) mo/c/r/v/ex=Dev "F:\\Laptop Docs" #
) po

List files, sorting first by type, and then by time-last-modified from oldest to newest.

) filestatus/sort=type+tlm

Now, with minimal uniqueness, reversing the sort order of the 'tlm', and choosing specific fields to display via the /fields= switch (/fi= for short). Here we just display the name and time last modified from newest to oldest. Fields can be in any order.

) fi/so=ty+-tlm/fi=na+tlm

Show all files bigger than a megabyte in, and under, your current directory, showing the file's directory. Sorted by size.

) fi/so=siz/fi=dir+na+tlm/sizegt=1000000 #
Filter switches available for size and times (created, modified, accessed). Test for =, !=, >, <, >=, <=

Where is that file? Find it on your 'Searchlist' (similar to a 'path')

) searchlist C:\utils,c:\Dev,c:\users\myid\documents
) path backup_dev.cli
c:\Dev\backup_dev.cli

How about expanding the contents of a file, and extracting data from the first line?

) string [batchrun.out]
) write Batch run was of type: [!string/start=1/end=4]
Batch run was of type: INV1
) rename batchrun.out batchrun_[!string/start=1/end=4].out

Here, the file named 'batchrun.out' had a first line of 'INV1 0000 23.4' and we put it in a string variable (un-named) by using the file expansion notation of square brackets, and then used the first 4 characters of it to display the type of the run, and renamed it also.

More Conditional logic in a sample macro file:

```
! Check to make sure we have 3 arguments.
[!nequal %num% 3]
  Write Error: Macro requires 3 arguments.
  Return
[!end]

Write Arguments 1 thru 3 are %1-3%
Write Argument 3 is %3%
! Set an integer variable named ‘var1’ to the value of the first argument.
Ivar/name=var1 %1%
! Set an integer variable named ‘var2’ to the value of the second argument.
Ivar/name=var2 %2%
! Now report if the sum of arg1 and arg2 equals arg3
[!iequal [!iadd [!ivar/name=var1] [!ivar/name=var2]] %3%]
   Write Argument 1 plus Argument 2 equals Argument 3
[!else]
   Write Argument 1 plus Argument 2 does not equal Argument 3
[!end]
```