HUBBARD
Software

CLI
(Command Line Interpreter)

Notice

This documentation may contain errors or omissions. It is provided strictly on a "best efforts" basis. The company takes no responsibility for damages that occur due to such inaccuracies. The company will attempt to make corrections in a timely fashion after being notified of any issues.

Trademarks

Microsoft – Is a registered trademark ® of Microsoft Corporation
Windows - Is a registered trademark ® of Microsoft Corporation
Oracle – Is a registered trademark ® of Oracle Corporation
Java – Is a registered trademark ® of Oracle Corporation

Overview

This manual describes the features and use of the Hubbard Software Command Line Interpreter (CLI). The CLI is a form of "Shell" in that it provides a text based command access to the features of the underlying Operating System (OS). The primary differentiator of this CLI over other shell type command handlers is that most of the commands are implemented directly in the CLI, and it is user extensible (in the paid-for versions). The CLI is written in Java® and all the commands and some other features are dynamically loaded, and driven off property files. Thus it is highly customizable, and can be easily internationalized.

Section 1 Describes the general command line interface.

Section 2 Describes the commands themselves.

Section 3 Describes the Pseudomacros.

Section 4 Describes how to customize the CLI at startup and change various defaults.

Section 5 Describes how to add a command or psuedomacro yourself.

Typography

For the command reference, arguments and other descriptive text may be referenced in italics, within square brackets for optional information, or angle brackets for descriptive text.

Examples:

) write *[optional text argument(s) to print]*

This is not to be confused with literals contained within square brackets, which have a special meaning to the CLI as described in Section 1.

And:

<field to use> - Where the exact details of the field are described further on in the manual.

Table of Contents

Section 1 – Using the CLI

Overview

The CLI is, as its name implies, a "Command Line Interpreter. It is a text based interface where you enter commands at a prompt, and get text results back. On many operating systems this is sometimes called a "Shell". These commands are intended to give you control of Operating System functions, such as running other programs, listing and manipulating disk files, and the like. "CLI" is a generic term for these kinds of programs that provide text based access to an Operating System. Many vendors have some version of a CLI, which may provide similar functions, but with a different syntax than you see in this program.

The Hubbard Software CLI is unique in that it is extensible and customizable by the user (In the advanced versions). You can even write your own commands if you are proficient in the Java language. It also has a number of powerful features not typically found in other shell programs, such as inline functions (called Pseudomacros) and advanced built-in capabilities where other shells require external programs to do things like string manipulation, math, and advanced file operations.

Important Note

The CLI is very customizable. The manual here will use the default settings for all examples. Be aware that in the advanced versions, it is possible to change virtually all the text you see, or enter. You can even change the command names themselves.

Running the CLI

The CLI is a program written in the Java programming language. As such, it is shipped as a "Java Archive" or "JAR" file. Specifically it is an "Executable JAR file". Java is very portable, and thus the CLI should run on almost any computer, running almost any Operating System (OS), as long as Java runtime support is installed. You can start the CLI in any way that your OS would normally start the Java runtime. For example, on a Microsoft® Windows® OS, you could start it from the 'DOS Command Prompt' (cmd.exe) with something like this:

```
C:\Users\Dave> java –jar CLI.jar
```

Where 'java' is the name of the java runtime, '-jar' is a switch to the runtime saying you want to run an executable jar file, and 'CLI.jar' is the name of the CLI itself. It is beyond the scope of this manual to list all the possible ways of starting the CLI. But one suggestion would be to create a 'Shortcut' that runs it directly by clicking on an icon on your desktop.

It is possible to pass further switches to the CLI itself. See Section 4 on how to customize the CLI's startup behavior this way.

You can also specify a filename at startup which is treated like a "CLI Macro" and is immediately run by the CLI when it starts. A Macro is a file that contains one or more commands you want executed in a group. Much like the Windows/DOS .BAT (Batch) files. More on Macros below.

Basic Concepts

The CLI gives you a default prompt for input which is a single closed parenthesis followed by a space. As an example, we will use the simple 'WRITE' command, which merely echoes back the text you pass it as an argument.

) write Hello World!
Hello World!

Note in our examples, we will use the **BOLD** font for the output from the CLI, and regular text for input you enter.

Text Entry

The CLI accepts text in plain form or within double quotes. You might use double quotes if you don't want the CLI to interpret special characters such as a semicolon ';'. NOTE: In double quotes, a slash character, either '/' or '\', is an ESCAPE character. On your operating system it will be the reverse of the one used in directory paths, so that you are not 'escaping' characters all the time if you cut and paste paths. The escape character tells the parser to just accept the next character as is. Like if you actually want a double quote to be in the string. If you want to actually have the escape character, just use two of them, like '//' or '\\'.

Comments

Any line starting with an exclamation point '!' is treated as a comment line and is ignored. This is particularly useful in 'macros'. See below.

Arguments

Commands and Pseudomacros (see below) may accept arguments. Arguments are separated by either spaces, tabs, or commas. In the above "Hello World" example, there are two arguments to the 'write' command: 'Hello' and 'World!' . There are also delimiters saved in between them.

Note that various commands and pseudomacros may take numeric values. All arguments are in fact simply text strings, so any numeric text is simply converted as needed. This makes the CLI 'loosely typed'.

Prompt and Prefix

When you are prompted for text entry, you are actually seeing two things:

- Optional 'Prompt' values
  The 'Prompt' is not enabled by default. However it is simply a list of commands with no arguments, whose output you will see. A useful example would be the TIME command for example, where the time would be displayed on the line before the prefix. This can be set by the use using the 'PROMPT' command. See its description later in the manual.

- A 'Prefix'
  This is the closed Parenthesis and a space by default. But can be changed. See the description of the 'PREFIX' command later in the manual.

Minimal Uniqueness and Case Sensitivity

All commands, switches passed to them, and many values for switches, can be abbreviated. They can be shortened to the point that they can be distinguished from some other command or switch. So the previous example could simply have been:

) w Hello World!
Hello World!

Because the 'w' is actually minimally unique. No other command starts with a 'w'.

Also, all commands and switches are case-insensitive. So you could also have done this:

) WriTe Hello World!
Hello World!

If you really felt like it.

Switches

Many commands allow 'switches' which modify their behavior in some way. Virtually all commands accept the **/list=** switch, which redirects the output of the command to a file, just for the one command. A switch looks like this:

) write/list=test.txt Hello World!

The output would now appear in a file named 'test.txt' in your working folder.

As mentioned, switches can be minimally unique and are case-insensitive, so the following would work:

) write/L=test.txt Hello World!

Some switches require values after an '=', like the above, but some do not. It depends on their function.

The common switches across all commands are:

/list= - Redirect output to a file for the one command.
/1= - Set the error action for Class1 exceptions for just the one command. See section on error handling below.
/2= - Set the error action for Class2 exceptions for just the one command. See section on error handling below.

Pseudomacros and File Expansion

A **'Pseudomacro'** is a built in function that can be used anywhere in a command line, and are expanded to a value before the command is executed. There are a variety of them that can do useful things. Section 3 details all of them. But briefly, as an example:

) write It is now [!time] on [!date]
It is now 16:56:17 on 08/30/15

You can see how the current time and date are displayed. All Pseudomacros start with '[!' and end with ']'. They can also be nested to any depth if it makes sense. The Pseudomacro is run before the command is, as part of the command line expansion, and the resulting text is inserted as if typed in the command line.

There is a very special class of Pseudomacros, which allow programmatic control flow. These provide an IF/THEN/ELSE like structure, and are given their own subsection later in this section.

'File Expansion' is a notation where you can put a filename in square brackets and the CLI will expand the contents of the file into your command line. Note that it will only expand the first line of the file, although if the line ends with an ampersand '&' character, it signals a continuation line will follow, and the following line will be appended to the first, for as many lines as you want until some line does not end in an ampersand.

Example:

A file named 'myfile.txt' has one line with the text "Hello World" in it.

) write My message is: [myfile.txt]
My message is: Hello World

User Environment

The CLI maintains a number of environmental settings, most of which the user can change. The classic example is your current working directory, sometimes called a 'cwd' for short. In the CLI it is simply called 'DIRECTORY'. There are commands and Pseudomacros to allow you to set, display, and use these values. Specifically, they are (See commands and Pseudomacros of the same names):

- PROMPT – The commands to be run with each user prompt.
- PREFIX – The text right before the user's text entry spot. Default ') '.
- DIRECTORY – Current working directory for file operations.
- SEARCHLIST – List of directories that are searched for files. Like a 'path', but more generic. For example, the 'TYPE' command will look in the current directory for the file, and if not found there, will look in all the searchlist directories to find it.
- STRING – A variable that stores text values. These can be named, providing an unlimited number of them.
- IVAR – A variable that stores integer values. Values are signed 64 bit values. These can be named, providing an unlimited number of them.
- FVAR – A variable that stores floating point values. IEEE 'Double' values. These can be named, providing an unlimited number of them.
- CLASS1 – The current error handling action for Class 1 errors. See below for more detail on structured error handling.
- CLASS2 – The current error handling action for Class 2 errors.
- LISTFILE – The current output file for CLI output. 'System.out' and 'stdout' are synonymous and are the default, which is your console usually.
- LOGFILE – The current file where all CLI activity is being logged. Usually disabled.
- SYSENV – The System Environment set of values. See the SYSENV command.

The Environment Stack

All the above environment settings are stored on a 'Stack'. There is a 'PUSH' command to allow you to add another level to the stack, and a 'POP' to return to a previous level. The 'top' of the stack is the current working set of these. However a few commands allow you (through a switch) to access the previous level setting. This allows you to make temporary changes, and then 'pop' back to the previous settings. When you 'push' a level, all of these are copied to the new level. When you 'pop', the previous level values are restored. See the 'PUSH' and 'POP' commands for details.

Error Handling

There are two 'exception severities' which can occur. They are known as 'Class 1' and 'Class 2' exceptions. Basically, Class 1 errors are those for which the command or other operation cannot realistically continue. Class 2 is more like a warning, and the function may try and continue. These are different from the 'error code' which is returned. Any error causes an 'exception' of one of the two classes. And for any given error code (A numeric value printed with the error message), the exception severity may differ from command to command. So for example, a 'File does not exist' (Error code 10035) may be a serious error in one command, but not in another.

For each exception severity there is an action level that the CLI will take. These actions are:

ABORT – Exit the CLI program. The CLI exits to the OS returning the error code that triggered the event.
ERROR – Print an error message and abort the command, and also abort a macro if the command is in a macro. Also aborts subsequent commands on the same command line.
WARNING – Print a warning message and try and continue as best it can. Possibly with some default.
IGNORE – Suppress any message to the user and just continue if possible.

The default values for the classes are:

Class 1 – ERROR
Class 2 – WARNING

There can be a couple of very serious internal errors which might abort the CLI, but there is no way the CLI could continue operating in these cases. An ABORT is unconditional in those, such as might be caused by a hardware failure.

There may be cases where you want to try and override the behavior of the CLI when one of these occurs. Like continuing an operation even if a file does not exist for some reason.

You can set/display these with the 'CLASS1' and 'CLASS2' commands and Pseudomacros.

Also, all commands accept a **/1=** and **/2=** switch which lets you set the action just for the duration of the command. So for example, if you don't want a macro to abort if you are just making sure that a temp file has been cleaned up, but it might not exist, you could do:

) delete/1=ignore temp_work_file.dat

And if the file isn't there, no big deal.

Special Command Shortcuts – Angle brackets and Parentheses

A powerful feature of the CLI is use of argument repetition groups in angle brackets, and command repetition groups in parentheses.

Angle Brackets:

A list of text items within angle brackets is expanded by concatenating each item to any adjacent text, and expanding it into individual items on the command line before it executes. These can be nested to any depth.

Best to have an example. Using the 'TYPE' command that simply displays a file:

) type data.txt
LINE 1 of the file
LINE 2 of the file
LINE 3 of the file

Simple enough. But if you wanted to type a couple of files with similar names, say purchase order data (in PO_data.txt) and invoice data (in INV_data.txt):

) type <PO INV>_data.txt
LINE 1 of the PO file
LINE 2 of the PO file
LINE 3 of the PO file
LINE 1 of the INV file
LINE 2 of the INV file

You see that both a PO_data.txt and INV_data.txt file were displayed. You can use this in any command, so you could delete both files for example. Or move them.

This notation can be nested, and you can include Pseudomacros in it which will be expanded first.

If you have more than one group, the CLI iterates through the left most group the slowest, and the right most the fastest. Best with an example:

) write <a b c>_<x y>
a_x a_y b_x b_y c_x c_y

Note 'a' is glued to 'x' and 'y' with the underscore (or whatever text) in between, then 'b' is handled the same, then 'c'. Try it. Try 3 or 4 groups.

Parentheses

Text within parentheses cause the command to be repeated. There is also a special case where you can start the command line with a list of commands within parentheses, and each command will run against the same set of arguments. So a multiple command example:

) (delete create) myfile.dat

The above will delete 'myfile.dat' if it exists, and then create a new, empty, file of the same name, in the current working directory.

A simple example without concatenation:

) wr A (1 2) B
A 1 B
A 2 B

Note how the command repeated for each value in the parentheses.

An Example with multiple arguments, contained within parentheses:

) write (a b c)_(x y)
a_x
b_y
c_

Note how the command is actually repeated, unlike the angle brackets where arguments are expanded.

Also note how the arguments are used. The CLI tries to combine the first element in each left group to the corresponding element in the right group, until the right group runs out, in which case it continues with just nothing appended.

When combined with angle brackets, Pseudomacros (like String variables), and the like, this notation becomes very powerful.

Macros

A macro is simply a file containing CLI commands. It is a powerful way to implement useful operations in a group. So it's kind of liking writing your own command, or CLI subroutine.

Macro execution:

When you type in a 'command' at the prompt, the CLI first tries to run a built in CLI command that matches it. However, if no command matches the text you type in, the CLI will attempt to find a file by that name and run commands in it.

The CLI will first look in your current working directory. If it is not found, it will look in the directories on your Searchlist (See SEARCHLIST command).

If it is still not found, it will look for a file with your text and ".cli" appended. It is good practice, but not required, that you name your macros <my name>.cli .

The CLI then runs the commands in the macro, although you will note that the CLI does not echo the commands themselves (unless you turn on 'trace mode', see the TRACE command), nor do you get prompts for each command. You only get the output.

See also the RETURN command that returns from a macro.

Note that there is no problem with a macro calling another macro, to any depth you want.

Macro Arguments:

Just like with a command, you can provide arguments to your macro, and access them from inside it. The notation is to surround the reference with percent signs. The legal references are:

%num% : The number of arguments. Includes the 0^{th} (see below).
%n% : The nth argument.
%m-n,i% : Every ith arg from m to n.
%-n,i% : Every ith arg from 0 to n.
%m-,i% : Every ith arg from m to the last.
%-,i% : Every ith arg from all the args.
%m-n% : All args from m to n.
%-n% : All args from 0 to the nth.
%m-% : All args from m to the end.
%-% : All arguments.

Note that 'Argument 0' is the macro name itself. The real arguments start at 1.

Macro Example:

Say you have a file named ‘mymacro.cli’ in your working directory that has the following 3 lines in it:

write Number of args passed is %num%
write Argument 1 is %1%
write Arguments 2 through 4 are %2-4%

You can call it as follows:

) mymacro A B C D
Number of args passed is 5
Argument 1 is A
Arguments 2 through 4 are B C D

You will get an error if you try and reference a non-existent argument.

Conditional Execution – Real Power!

You can control both the expansion of your command line, and the execution of commands using some special 'pseudomacros'. These are actually built-in control structures.

There are 3 special cases of these:

- Test Pseudomacros – Like [!equal a b]
- The [!else] pseudomacro – Just like in regular programming.
- The [!end] pseudomacro – Ends a conditional block.

Test Pseudomacros:

There are many of these available, but they basically boil down to tests against 3 types of data, and for each type, the usual set of comparisons. The name of the pseudomacro you want is a single character representing the data type (see below), followed by the name of the operator. These, as with all other pseudomacros, can be minimally unique.

Note: All String type test pseudomacros support the **/ignorecase** switch for case insensitivity.

Types:

String – Start with 's'
Integer – Start with 'i'
Floating Point – Start with 'f'

Comparisons:

Equal – 'equal' (or minimally unique can be 'eq')
Not Equal – 'nequal' (or minimally unique can be 'ne')
Greater Than – 'gt'
Greater Than or Equal – 'ge'
Less Than – 'lt'
Less Than or Equal – 'le'

Because comparing two strings is the most common there are two shortcuts where you don't need the data type prefix.: [!equal] is the same as [!sequal] , and [!nequal] is the same as [!snequal] . All other comparisons require the data type.

Note that the reason for the numeric tests are so that, as an example with Integers, '00001' is equal to '1' as an Integer comparison [!ieq], but not as a String comparison [!eq] or [!seq].

Block Pseudomacros

After the test, you are in a conditional block of the command line or commands. You end the block with [!end]. You can also have an 'else' case with [!else].

Example within a command line:

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

Now of course this example doesn't make much sense, but combine it with macro arguments and you can do interesting things! Like :

write %1% and %2% are [!equal %1% %2%]Equal[!else]Not Equal[!end]

Example with multiple lines:

You can execute, or not, whole groups of lines, which is very powerful in macros. You can have a macro with the following lines:

[!igt %1% %2%]
Write %1% is Greater than %2%
[!else]
Write %1% must be less than or equal to %2%
[!end]

The [!igt] here means 'Integer Greater Than'.

Note you do not need the [!else] block if you don't want it.

Special note:

If you do not close a control structure, like forget the [!end], the CLI returns to the prompt with a special prefix to let you know it is still in a control block.

If you are in a non-executing section, your prefix will have a '\' before it. If you are in an executing section, it will have a '!' before it. See the following example:

) [!equal a b]
\) wr hello
\) [!else]
!) wr bye
bye
!) [!end]
)

You see that since 'a' does not equal 'b', the '\)' prefix means that the commands you enter just get 'eaten' and not executes, but after you type in [!else], you are now in an executing block with '!)', until you end the block with [!end]. This will occur if you don't end a macro too, when you return to the command prompt it will look like one of these to tip you off.

Section 2 – The Commands

This section will describe each of the commands available to the user. Refer to Section 1 for details on the command line structure and some core features and limitations available through some of the commands.

Command Line Structure

A command line may contain one or more commands, separated by a 'command separator'. By default this is the semicolon ';' however this is a customizable character. See Section 4 on how to customize this.

A simple command is the 'WRITE' command. It merely echoes the text it is given. An example of a command line would be:

) write Hello World!

Note that the parenthesis ')' is the prefix used to prompt the user, as described in more detail in Section 1.

Another example might be:

) wr Hi There!;wr More Greetings!

Here, we used minimal uniqueness for the 'write' command, abbreviating it to just 'wr', and also have two commands on the line. Both will execute.

IMPORTANT NOTE: If a command on a multi command line has a serious error (Class 1), the subsequent commands will not execute.

The rest of this section is a reference for each command, an alphabetical order.

NOTE ON SWITCHES: Virtually all commands will honor the /list=, /1=, and /2= switches as described in Section 1. They are therefore not repeated for each command below.

? – Shortcut for the ‘HELP’ Command

The ‘?’ is merely a shortcut for the HELP command. See the HELP command for more information.

BYE – Exiting the CLI

This exits the Command Line Interpreter.

Operation

This simply exits the CLI program, and returns an integer error code to the underlying Operating System. The default is a code of zero. Or you can specify a value to be returned.

NOTE: This operation is unconditional. It will exit, even if you pass an illegal integer value, in which case the value returned to the OS is undefined, but will be greater than zero.

Argument

One optional integer exit code to be returned to the Operating System. If none, zero is returned.

Examples

) bye *1*

) bye

CLASS1 – Set or Display the 'Class 1' error handling action

As discussed in Section 1, exceptions have two possible severities, a Class 1, or Class 2. You can override the default treatment of Class 1 exceptions with this command.

Argument

None – If no argument is present, this reports the current handling.

The other possible argument values are (minimal uniqueness allowed):

Ignore – Do not report any Class 1 exceptions, and try and proceed with command processing as best as possible. Not recommended unless you really are expecting an error and don't care.
Warning – Report a 'Warning' message and try and continue as best as possible.
Error – Report an 'Error' message, abort the current processing. This will also break out of a macro. It will also stop executing the current command as well as any others on the same command line. This is the default for Class 1 exceptions.
Abort – Report an 'Abort' message and exit the CLI. The CLI will exit with a non-zero error code.

Example

) write [!string xyz]
Error (10024) : Pseudomacro does not accept arguments.
) class1 ignore
) write [!string xyz]

)

Note that the [!string] pseudomacro does not accept an argument in this mode of operation, and thus returns an error, indicating a 'Class 1' exception. After setting class 1 to 'ignore' you get no message, and the write command attempted to continue, with a blank string.

CLASS2 – Set or Display the 'Class 2' error handling action

As discussed in Section 1, exceptions have two possible severities, a Class 1, or Class 2. You can override the default treatment of Class 2 exceptions with this command.

Argument

None – If no argument is present, this reports the current handling.

The other possible argument values are (minimal uniqueness allowed):

Ignore – Do not report any Class 2 exceptions, and try and proceed with command processing as best as possible.
Warning – Report a 'Warning' message and try and continue as best as possible. This is the default setting.
Error – Report an 'Error' message, abort the current processing. This will also break out of a macro. It will also stop executing the current command as well as any others on the same command line.
Abort – Report an 'Abort' message and exit the CLI. The CLI will exit with a non-zero error code.

Example

```
) type junk.txt
Warning (10035) : File '.\junk.txt' does not exist.
) class2 ignore
) type junk.txt
)
```

Note that the 'type' command returns a warning by default if the file does not exist, indicating a 'Class 2' exception. Setting it to 'ignore' causes no message the second time.

COPY – Copies one or more files to another

Copies one or more source files to another destination file, optionally appending them to an existing file.

Arguments

At least two.

First Argument – The destination file. It can exist or not in the current working directory. The SEARCHLIST is not searched for the file to see if it exists. See switches.

Subsequent arguments – Source files or templates. The SEARCHLIST is not searched for the file.

Switches

/delete – Will delete the destination file if it exists. No error if it does not exist.
/append – Will append to an existing file. No effect if it did not exist.
/verbose – Will cause detailed reporting of the files copied to the destination.
/exclude= - Specifies a list of file templates to exclude from the operation. Accepts more than one if you put them in double quotes and separate them with the 'path separator' (Like ';' in Windows or ':' in Unix). Matches the filename only.

Operation

If the destination file does not exist, it will be created with as many of the attributes of the first source file as possible. Thus, if you only have one source file, this will make a true copy of it.

Subsequent files are always appended to the destination after the first one. Thus, the /append file only applies to the first source file copied.

You can use any template for the source file(s) that you could use in the FILESTATUS command.

Example

) copy allsource.txt *.java
) copy allsource.txt *.java
Error (10059) : File 'allsource.txt' already exists.
) copy/del allsource.txt *.java
) copy/app allsource.txt ..\Converter.java
) copy/exclude=*.bak allsource.txt *.*

CREATE – Create a file or directory

Use this command to create new files or directories. For a file, there is an option to enter text into it.

Arguments

A list of one or more names to be created. These can be relative to the current working directory, or absolute paths to other directories.

The file or directory named cannot already exist.

Note: If you are creating a directory, any missing directories in the path will also be created. So you can create a whole sub-directory structure in one shot.

Switches

/directory – Tells the command that a directory is to be created instead of a file.
/input – For a file, the user will be prompted with a ‘)) ’ for input to put into the file. You end your input with a single ‘)’ on a line.

An error will be reported if you try and use both switches, as you cannot put data in a ‘directory file’ itself.

Examples

) cre foo.txt
) cre/inp foo2.txt
)) Some text I want in the file.
)) Another line of text.
)))
) type foo2.txt
Some text I want in the file.
Another line of text.
)

DATE – Display the system date

Display the current system date using either the default format, or using one you provide via a switch.

Arguments

None.

Switches

/format = Specifies a format to use for displaying the date. This format must be in the same format as the Java java.text.SimpleDateFormat class definition. This overrides the CLI default format. See Section 4 on customization for a way to change the default date format.
/tz= Specifies a timezone ID to use. Must be from the set displayed using the TIME command's /showalltimezones switch and is case sensitive. May also be from any timezone aliases specified in the resource property file.

Note: You may also include time fields in your format.

Examples

Note minimal uniqueness of switch in one of the examples.

) date
09/05/15
) date/format=yyyy-MM-dd
2015-09-05
) date/f=yyy-MM-dd:HH:mm:ss
2015-09-05:21:33:25
)

See also the [!date] pseudomacro which works much the same way.

DEBUG – Set or Display the current debug setting

This command is reserved for developers working on the CLI program, and is not guaranteed to do anything specific.

Currently it will display the setting if no argument is passed in, or accepts ‘on’ or ‘off’.

The behavior of the CLI if this is turned ‘on’ may vary. It may produce additional tracing or error detail for example.

DELETE – Delete's files or directories

This command will delete files or directories. It has options to print the name before deletion, and can also prompt the user for every file in the event a template is used. It can also recurse through directories, deleting all files in a directory tree.

Arguments

One or more names of files or directories, or file templates such as '*.bak' .

Switches

/confirm – Causes the CLI to ask for confirmation before deleting each file or directory. The positive answer is 'y' or 'yes'. Anything else is a negative answer.
/verbose – The CLI will print a line with the name of each file or directory before attempting to delete it.
/recurse – For directories, this will recursively empty a directory structure under a directory before deleting the directory itself.
/exclude= - Specifies a list of file templates to exclude from the operation. Accepts more than one if you put them in double quotes and separate them with the 'path separator' (Like ';' in Windows or ':' in Unix). Matches the filename only.

Note: If a directory is not empty, you will get an error unless you used the /recurse switch.

Examples

) del foo.txt
) del/verb foo2.txt
Deleting File: C:\Users\Marvin\Documents\Dev\workspace\CLI\foo2.txt
) del/v foo2.txt
Deleting File: C:\Users\Dave_2\Documents\Dev\workspace\CLI\foo2.txt
Error (10035) : File 'foo2.txt' does not exist.
) del/confirm foo3.txt
Delete 'C:\Users\Dave_2\Documents\Dev\Test\foo3.txt' ? y
) del/excl="*.java;*.c" *

As with all commands, all the switches can be minimally unique. So '/c' will work for '/confirm', etc…

Note the /exclude example will delete all files in the folder except files ending in .java or .c .

DIRECTORY – Set or Display the current working directory

Change your working directory, or just display it.
Also see the CD command which is a synonym for this command.

Argument

Zero or one argument. With no argument, the command displays the current working directory. With an argument, it changes the working directory to the one specified by the argument.

Note: You can use 'relative' motion from the current directory. So for example, you can use the common '..' notation to move up a directory level, to the parent of the current working directory. You can use this in any combination, such as '..\..\test'. Or just the name of a folder or path under your current directory, such as 'testfiles' or 'testfiles\reports'. Or you can specify an 'absolute' path from your root, such as (in Windows) 'c:\users\arthur\documents\dev'.

Switches

/initial – Display the value it was when the CLI initially started up.
/previous – Display the value of the previous environment stack level. Will cause an error if there has not been a 'push' command to cause there to be a 'previous' level.

Note: You will get an error if you try and use both switches together, or with an argument.

Use Note: If you want to change to the 'initial' or 'previous' directory, use the Pseudomacro form of this command as an argument. For example, to change to the initial directory, do this:

```
) dir [!dir/i]
```

EXECUTE – Run an external program

This command allows you to run other programs from the CLI, and optionally redirect their input, output, and error streams.

Note: This is synonymous with the XEQ command, which is a shortcut as "X" is minimally unique for it.

Arguments

The first argument is the program name. Depending on the Operating System, you may be able to omit the file extension. For example, in Windows, you can omit the ".exe" extension.

All other arguments are passed to the program.

By default, regular console output and error output go to your console. However you can redirect those using switches. The same is true for the program's standard input it might read on.

Switches

/string and /string= - Causes the return value of the program, usually an integer on most Operating Systems, to be stored in a String variable for your use. With no value (/string), it goes in the current default String (with no name). With a value (/string=<name>) it goes in the named String variable.

/output= - Redirects the program's stdout to the file name you specify. If the file does not exist, it will be created. If it does exist, it will be re-created (deleted and created).

/error= - Redirects the program's stderr to the file name you specify. If the file does not exist, it will be created. If it does exist, it will be re-created (deleted and created).

/input= - Redirects the program's stdin to the file name you specify. If the file does not exist, it will be created. If it does exist, it will be re-created (deleted and created).

Examples

) execute java myjavaprog
) execute/string java myjavaprog
) string
0
) exec/str/out=myjavaprog.out java myjavaprog
) type myjavaprog.out
My Java Test Program: Hello World!
) string
0
)

FILESTATUS – List files in the file system

Lists files in the underlying file system, allowing various filters.

Syntax:

) filestatus*[optional switches] [optional file/path template(s)]*

Operation:

The command lists files according to the switches and arguments, if any. It is limited by the underlying Operating System, so some fields of data about a file may not apply in all cases. Also, the 'template characters' which are used as 'wildcards' to match text may not be the same as that of some other shell you may be used to. See below for more detail.

Default behavior:

If there are no switches or arguments passed to the command, it will show a file listing of the current working directory. (See the 'DIRECTORY' command). It will also display a fairly complete set of file information. The specific fields are described below.

Switches:

tlmeq=	tlagt=	tcrge=	sizelt=
tlmgt=	tlage=	tcrlt=	sizele=
tlmge=	tlalt=	tcrle=	sort=
tlmlt=	tlale=	sizeeq=	hidden
tlmle=	tcrgt=	sizegt=	noheader
tlaeq=	tcreq=	sizege=	fields=

Also:

/exclude= - Specifies a list of file templates to exclude from the operation. Accepts more than one if you put them in double quotes and separate them with the 'path separator' (Like ';' in Windows or ':' in Unix). Matches the filename only.

This command has one of the most sophisticated set of switches of any command. Therefore we will break them out into the major functional groups here. There are a few primary groups:

- The '/hidden' switch for seeing Hidden files
- Date/time and Size filters
- The '/sort=' switch for sorting your output
- Formatting control: The '/fields=' switch

The Hidden File Filter - /hidden switch

Normally files which the operating system has marked as 'hidden' do not show up on the listing. If you want to see these, use the '/hidden' switch.

Date/Time and Size Range Filters

You can filter the list of files based on ranges of date/time or size fields. For example you can list just files modified before last month, or larger than some number of bytes. It is important to note that the filters are ANDed together, which allows ranges, such as 'greater than the first of last month, AND less than the first of this month' which of course gives you all the files modified last month. Some combinations may simply make no sense of course, but that is up to you!

USE NOTE: A common date-range mistake is to do 'greater than date a, and less than date b'. This would not pick up a file created exactly ON 'date a'! And it is certainly possible for a file to be created by some overnight job exactly at midnight! (Personal experience here!)

So to show all files from last month:

Bad: 'greater than the first of last month, and less than the first of this month'
Better: 'greater than or equal to the first of last month, and less than the first of this month'

USE NOTE: Be careful of using 'equals' alone with date/times. Some operating systems maintain times smaller than the seconds that are displayed. So 13:15:23 might really be 13:15:23.06 which is of course not equal.

Filter Switches

All these switches have names that start with the field you want compared, followed by a 2 character comparison operator code.

Format of switch names: ***/<fieldname><operator name>=<date/time value>***

Fields:
TLM – Time Last Modified
TLA – Time Last Accessed
TCR – Time Created
SIZE – Size of the file in bytes

Operators:
EQ – Equal
GT – Greater Than
GE – Greater Than or Equal to
LT – Less Than
LE – Less Than or Equal to

Value for date/times:

The value should be as 'yyyy-MM-dd:HH:mm:ss' with the time parts being optional (you can just drop the seconds, seconds and minutes, or the whole time stamp).
NOTE: This format is the default, but is configurable at startup. Also, the notation here is from the Java language as used in the java.text.SimpleDateFormat class. So for example, the upper case 'HH' means hours using a 24 hour clock. One PM being = 13.

Examples

) filestatus/tcrge=2015-07-01

Shows all files in the current working directory that were created (TCR) on or after July 1, 2015 (Greater than or Equal to).

) fi/tlmge=2015-07-01/tlmlt=2015-08-01/sizege=1000000

Uses the minimally unique command name 'fi', and shows all files in the working directory that were modified (TLM) any time during July 2015 and are equal to or larger than one million bytes.

The Sort Switch - /sort=

The default sort order is the directory the file is in, and then the filename, both in ascending order. The /sort= switch allows you to override that order for a variety of fields. You specify the fields as the value of the switch, separating them with + signs. You can put an optional minus sign (hyphen, '-') before the name of the field to cause a descending sort order for that file. All text comparisons are done case-insensitive.

Field Names (Can be minimally unique, such as 'ty' for 'type'):

Path – Use the full canonical path of the file for the sort, including all the way up to its parent directory.
Name – The filename itself.
Size – The size of the file in bytes.
Directory – The directory the file is in.
TLM – The 'Time Last Modified' of the file. (Ascending is oldest to newest.)
TLA – The 'Time Last Accessed' of the file. (Ascending is oldest to newest.)
TCR – The 'Time Created' of the file. (Ascending is oldest to newest.)
TYPE – The type, currently either a FILE, DIRECTORY, LINK, or OTHER. (Sorted in alphabetical order)
Examples:

) fi/sort=size+-tlm

Shows all files in the current working directory, sorted by size smallest to largest, and if there is a tie on size, then by time last modified, newest at the top, oldest at the bottom.

) fi/so=type+tcr

Shows all files in the working directory, with sub-directories at the top, files at the bottom, and then in order of time created from oldest at the top to newest at the bottom.

Formatting Control

/noheader – Accepts no value, and simply causes the omission of descriptive headers.

/fields= - Allows you to specify which fields are displayed. Note that there is a default set, which can be varied with startup parameters (See Section 4).

> The field names passed as the value of the switch are separated by the '+' character. The names can be minimally unique, and are:
>
> NAME – The filename.
> DIRECTORY – The directory the file (or sub-folder) is in.
> TYPE – Can be FILE, DIR, LINK, or OTHER
> SIZE – The file size in bytes.
> FLAGS – Can be 'E' (Execute access) and/or 'R' (Read access) and/or 'W' (Write access), and/or 'H' (Hidden).
> TCR – Time created.
> TLM – Time last modified.
> TLA – Time last accessed.
>
> Example:
>
>) fi/fields=na+ty+si+tlm
>
> Display the files' name, type, size, and time last modified. Note the minimally unique field names. You could even do this with:
>
>) fi/f=n+ty+s+tlm
>
> Because 'f' is minimally unique as a switch, as are 'n' and 's' as field values.

Filename paths and templates

The default directory that is listed is the current working directory (See the 'DIRECTORY' command). However you can list any folder you have access to, or even a full set of sub folders. You can also specify 'wildcard' template matching characters to filter the filenames.

Wildcards

* - The asterisk matches any number of characters.
? – The question mark matches any single character.
- The Hash mark (Pound sign) matches any number of characters AND all sub-directories under the point at which it is found in a template.

These are best explained with examples:

) fi #

Will show all files in the current working folder AND all sub-folders.

) fi #*.txt

Will show all files in the current and sub-directories that end in '.txt'.

) fi C:\users\marvin\dev\#*.java

This will show all files ending in '.java' that are in Marvin's 'dev' folder or below.

) fi/sor=-size #

Show all files in your working directory and sub-directories, sorted by size descending. Good way to find big files.

Root directory note: A path starting with simply the separator character like '\' on Windows will be interpreted as the root of the user's current file system that their current working directory is in.

Example (Windows):

\users\zaphod

Is equal to 'C:\users\zaphod' if the current working directory is also in the 'C:\' tree.

FVAR – Set or Display Floating Point Variables

As mentioned in Section 1, the CLI allows any number of variables. This command allows you to set or display the value of Floating Point numeric variables.

These are stored internally as a 'double' which is a 64 bit float in IEEE 754 format.

Also see the [!fvar] pseudomacro.

Argument

With none – Displays the variable.
With one – Sets the variable to that value.

Switches

/name= - Selects the name of the Floating Point variable to be other than the blank name. This can be any text value.
/previous – Display only. Displays the value of the variable as it was in the previous environment level after a 'push' command.

Examples

) fvar
0.0
) fvar/nam=fv1 123.45
) fvar/nam=fv1
123.45
)

HELP – Display various help text about the CLI commands

Displays helpful information about the CLI, commands, Pseudomacros, and miscellaneous topics.

Syntax:

) HELP <optional arguments>

Arguments

If there are no arguments, the command displays a brief listing of available topics. If there are arguments, they are topics for which you want more detailed information, and the command will display that for every argument.

Note: The topic arguments can be minimally unique. For example:

) help wr

Will show help text for the topic on the 'write' command because 'wr' is minimally unique for that command.

Switches

/commands – Will display a list of all commands with a brief one line description of each.
/pseudomacros - Will display a list of all pseudomacros with a brief one line description of each.

IVAR – Set or Display Integer Variables

As mentioned in Section 1, the CLI allows any number of variables. This command allows you to set or display the value of Integer numeric variables.

These are stored internally as a 'long' which is a 64 bit signed integer.

Also see the [!ivar] pseudomacro.

Argument

With none – Displays the variable.
With one – Sets the variable to that value.

Switches

/name= - Selects the name of the Integer variable to be other than the blank name. This can be any text value.
/previous – Display only. Displays the value of the variable as it was in the previous environment level after a 'push' command.

Examples

) ivar
0
) ivar/nam=var1 123
) ivar/nam=var1
123
) push
) ivar/nam=var1 2345
) ivar/nam=var1
2345
) ivar/prev/n=var1
123

Note: See how the previous value stack was still 123, even though the current value was 2345.

LEVEL – Display the current environment stack level

Simply displays the level of the user environment stack. The base level at startup is 1.

Also see the [!level] pseudomacro.

Arguments

None.

Switches

Just default set.

Examples

) lev
LEVEL 1
) push
) lev
LEVEL 2
) pop
) level
LEVEL 1
)

LISTFILE – Set or display the current CLI output file/destination

The LISTFILE user environment variable exists on the environment stack. This command allows you to display it or change it. It is where all CLI output goes.

Argument

If none, displays the current setting. Possibly modified by switches, see below.

One argument – Sets the current LISTFILE to the filename specified.

Note: For the initial startup value, you can use 'System.out' or 'stdout'. They are synonymous, and are typically the user's console for most Operating Systems.

Switches

/initial – Causes display of the initial setting. No argument allowed.
/previous – Causes display of the previous setting on the environment stack. No argument allowed. Causes an error if there is no previous level on the stack (I.E. no 'PUSH' command has happened)

Operation

When setting the LISTFILE to a filename, the SEARCHLIST will NOT be searched for the file, and the current working directory will be used as the directory for the file, or if a full pathname is specified, then that will be used.

If the file exists, it will be appended to. If not, it will be created.

If it exists, it will also be checked to make sure it is not a directory, in which case an error is reported.

Once set, if the file is in your current working directory (you did not specify a full path), it will NOT change if you do a DIRECTORY or CD command. It will remain set to the directory you were in at the time.

Example

```
) wr Hello World!
Hello World!
) listfile list.txt
) wr Hello World!
) listfile stdout
) type list.txt
Hello World!
)
```

Note that after the listfile was set to “list.txt”, the ‘write’ command did not echo anything. Also note that we had to reset the listfile back to the initial “stdout” setting, otherwise the ‘TYPE’ command would also have gone to the “list.txt” file.

LOGFILE – Set or display the current CLI logging destination

The LOGFILE user environment variable exists on the environment stack. This command allows you to display it or change it.

The LOGFILE, when set, logs all CLI activity to the specified file. This includes all input, prompts, and output.

Argument

If none, displays the current setting. Possibly modified by switches, see below.

One argument – Sets the current LOGFILE to the filename specified. Can also be the text "null", which will disable logging.

Switches

/disable – Disables logging. No argument allowed.
/previous – Causes display of the previous setting on the environment stack. No argument allowed. Causes an error if there is no previous level on the stack (I.E. no 'PUSH' command has happened)

Operation

If logging is disabled, you will get a message to that effect in display mode, rather than a filename. Also see [!LOGFILE] which behaves a little differently.

When setting the LOGFILE to a filename, the SEARCHLIST will NOT be searched for the file, and the current working directory will be used as the directory for the file, or if a full pathname is specified, then that will be used.

If the argument is "null", logging will also be disabled, as when using the /disable switch. This allows you to use the pseudomacro form as an argument. Ex:) logfile [!logf/prev] Which will disable logging if the previous environment stack level also had it disabled.

If the file exists, it will be appended to. If not, it will be created. If it exists, it will also be checked to make sure it is not a directory, in which case an error is reported.

Once set, if the file is in your current working directory (you did not specify a full path), it will NOT change if you do a DIRECTORY or CD command. It will remain set to the directory you were in at the time.

MESSAGE – Display the text associated with message codes

Displays the text for a message code, such as error codes. Also see [!message] .

Arguments

Allows display for one or more arguments.

Switches

Just the standard set.

Example:

) mess 10024 10025
10024: Pseudomacro does not accept arguments.
10025: Named String '%s' not found.
)

MOVE – Move files from one place to another in the file system

Moves files from one place to another. Also allows for a 'copy' mode where it simply makes a new copy of the file(s) (Not to be confused with the COPY command). It will honor templates, much like the FILESTATUS command, as well as the same file filters that the FILESTATUS command has, such as the date/time and size filters. It can also do an 'update' by replacing existing files only if the source file has been modified more recently than the destination file.

Files retain as much attribute information as is possible with the destination file system. So things like 'Time Last Modified', 'Time Created', 'Time Last Accessed' and access rights are all kept, within the limits of the underlying Operating System and File System.

Arguments

First argument – The destination directory to move the file(s) to.
Additional argument(s) – One or more filenames or templates specifying the source files (or directories) to be moved.

Switches

/verbosity= – Print extra information about the progress of the move, for each file. Allows 3 possible values. If you do not specify the '=' and just use '/verbosity', the default value is 1:
0 – No reporting except for errors. Runs silently.
1 – Reports only files that have been moved. This is the default.
2 – Full reporting. Reports all files it will try and move, and then if it could or not.
/copy – Copy the files, instead of truly 'moving' them. Leaves the originals in place.
/delete – Unconditionally delete an existing file in the destination directory.
/recent – Overwrite the destination file if it exists and the new file is more recently modified.
/flat – Move all files into the destination directory, regardless of what their origin directory structure was.
/exclude= - Specifies a list of file templates to exclude from the operation. Accepts more than one if you put them in double quotes and separate them with the 'path separator' (Like ';' in Windows or ':' in Unix). Matches the filename only.
And: The full set of filter switches from the FILESTATUS command such as /tlmgt= , /sizege=, etc...

Operation

If a template is used for a source, then any directory structure below that point in the template will also be created in the destination to match, unless the /flat switch is present. So for example, if you specified a source of '#*.txt', the command would look for all files ending in '.txt', anywhere under your current working directory ('#' means that). If a file was found, say, in a subdirectory path 'dev\src\data', then that directory path would be created (if not there already) from the destination. So if the destination was, say, 'C:\users\eddie' then the command will make sure that 'C:\users\eddie\dev\src\data' exists, and would move a file found, say, 'control.txt' in there.

Examples

) move/v=2 g:\Dev *.java
Moving: C:\Users\Eddie\Documents\Dev\Test\Bob\src\RegexFilter.java --> G:\Dev\RegexFilter.java
Moved: C:\Users\Eddie\Documents\Dev\Test\Bob\src\RegexFilter.java
Moving: C:\Users\Eddie\Documents\Dev\Test\Bob\src\Test.java --> G:\Dev\Test.java
Moved: C:\Users\Eddie\Documents\Dev\Test\Bob\src\Test.java

Without the /v (or /verbosity) switch, this would have just reported that they moved.

Or now, using the /copy switch. Note we will do it twice. (Assumes destination is empty to start.

) move/v=2/copy g:\Dev *.java
Moving: C:\Users\Eddie\Documents\Dev\Test\Bob\src\RegexFilter.java --> G:\Dev\RegexFilter.java
Moved: C:\Users\Eddie\Documents\Dev\Test\Bob\src\RegexFilter.java
Moving: C:\Users\Eddie\Documents\Dev\Test\Bob\src\Test.java --> G:\Dev\Test.java
Moved: C:\Users\Eddie\Documents\Dev\Test\Bob\src\Test.java
) move/v=2/copy g:\Dev *.java
Copying: C:\Users\Dave_2\Documents\Dev\Test\Bob\src\RegexFilter.java --> G:\Dev\RegexFilter.java
Warning (10064) : Destination File 'G:\Dev\RegexFilter.java' already exists.
Copying: C:\Users\Dave_2\Documents\Dev\Test\Bob\src\Test.java --> G:\Dev\Test.java
Warning (10064) : Destination File 'G:\Dev\Test.java' already exists.

Above, you can see the files were still in the source directory the second time around. Also note that a 'file exists' is a 'Class 2' warning exception, so operation continues with the subsequent files.

Now add the /recent switch. Also note the /v switch (/verbosity) with no value.

) move/v/copy g:\Dev *.java
Copied: C:\Users\Eddie\Documents\Dev\Test\Bob\src\RegexFilter.java
Copied: C:\Users\Eddie\Documents\Dev\Test\Bob\src\Test.java

) move/v=2/copy/recent g:\Dev *.java
Copying: C:\Users\Dave_2\Documents\Dev\Test\Bob\src\RegexFilter.java --> G:\Dev\RegexFilter.java
Skipping: G:\Dev\RegexFilter.java
Copying: C:\Users\Dave_2\Documents\Dev\Test\Bob\src\Test.java --> G:\Dev\Test.java
Skipping: G:\Dev\Test.java

Here, you can see the files were skipped because they had just been copied and thus kept identical time-last-modified stamps. They were not more recent. You would not have seen these lines with the default verbosity (no /verbosity switch or just with no value).

) move/v/exclude="*.bak;*.tmp" G:\Dev Test*

Moves all files starting with 'Test' EXCEPT those ending in '.bak' or '.tmp'.

PATHNAME – Display the resolved full path to a file or directory

This simply prints the full path to the file or directory. It will find it on the searchlist if just a name is given. It will return an error if the file does not exist. See also the [!pathname] pseudomacro.

Also see [!pathname] .

Argument

Only one allowed, which is the filename of the file or directory you want the path for.

Switches

Only the standard set: /list=,/1=,/2=

Operation

You get a 'Class 2' warning by default if the file cannot be found.
Also note that templates are not allowed.

PID – Display the CLI's 'Process ID'

This displays the integer Process ID as assigned by the Operating System.

See also the [!PID] Pseudomacro.

Arguments

None.

Switches

Only the standard /list=, /1=, /2= .

Example

) pid
PID: 7234
)

PREFIX – Set or Display the prefix text used in prompting for commands

See Section 1 for a description of how the CLI prompts you for input. This command allows you to set/display the current 'Prefix' part, which is by default just a closed parenthesis followed by a space. Also see the PROMPT command.

Argument

If none, displays the current prefix text.
If one, sets the prefix to that value.

Switches

/initial – Sets the prefix to the initial setting. No argument allowed.

Example

) pref
Prefix: ') '
) pref "* "
***** wr See the new prefix?
See the new prefix?
***** pref/init
)

PROMPTCOMMANDS – Set or Display the commands to run when prompting for commands.

Sets or displays the commands that can be executed as part of the prompt for user input. See Section 1 for more information. These commands cannot accept arguments, so must simply be display type commands such as DIRECTORY, TIME, DATE, and the like with no args. The results of the commands are shown on the lines before the 'Prefix'.

Arguments

None – Displays the current set of commands to execute.
One or more – A list of commands to execute, without arguments.

Switches

/clear – Causes the list of commands to be cleared.
/previous – Displays the previous user environment stack level setting.

Example

) prompt

) prompt date time
09/19/15
17:11:53
) wr Hello!
Hello
09/19/15
17:12:10
) prompt/clear
)

PUSH – Push a level on the environment stack

See Section 1 for a discussion of the user environment stack. This command 'pushes' a level on the stack, copying forward all the various settings and variables. If you change any and then 'pop' back, the changes will be lost on the previous stack level is restored. This is handy for making temporary changes, as within macros and the like. See also the POP command.

Note: This includes your complete set of String, Integer, and Floating Point variables. Named or un-named.

Also see POP and LEVEL commands, and [!level] Pseudomacro.

Arguments

None.

Switches

Just standard set of /list=, /1=, /2= .

POP – Pop a level off the environment stack

See Section 1 for a discussion of the user environment stack. This command 'pops' a level off the stack, restoring all user environment settings, as well as user variables.

See also the PUSH and LEVEL commands, and the [!level] Psuedomacro.

Arguments

None.

Switches

Just standard set of /list=, /1=, /2= .

Example

) string String for first level
) push
) wr Copied forward string: [!string]
Copied forward string: String for first level
) string Changed for second level.
) string
Changed for second level.
) pop
) string
String for first level
)

RENAME – Renames a file or directory

Simply renames a file or directory from one name to another.

Arguments

Arg 1 – The source file or directory you want to rename.
Arg 2 – The new name.

Switch

/verbose – Causes a message to be printed before attempting the rename operation.

Operation

The command will check to make sure that the source file exists, and that the new name does not. You will get an error to that effect if needed.

The command does NOT check the searchlist for these files. It only works in the current working directory if these are relative references. If they are absolute references, it will rename them to the fully qualified path and filename.

RETURN – Return from a macro

This simply returns right away from a macro. Useful if you are testing for some case, like legal arguments or whatever, and do not want to continue in some case.

NOTE: This command does nothing in interactive mode.

Arguments

None.

Switches

None. Not even the /1=, /2=, or /l= .

SEARCHLIST – Set or Display the current Searchlist

See Section 1 for more detail on the 'Searchlist'. This command lets you display or set the current searchlist.

The searchlist is a user environment variable which is used by some commands to find files, and to locate macros, among other things. See specific commands for details of its use.

Also see [!searchlist] .

Arguments

If none – Displays the current searchlist.
One or more args – Each argument should be a directory reference which is added to the searchlist in order.

Switches

/previous – Display the searchlist from the previous user environment stack level. No arguments allowed. Reports an error if there is no previous level. See PUSH and POP commands.
/clear – Clears the searchlist.

STRING – Set or Display the value of String Variables

Allows you to set or display the value of String variables.

You can have any number of String type variables, made unique with names. If you do not specify a name using the /name= switch, there is one 'blank' named String that is used.

Note: For named strings, the names to not conflict with other variable types such as Integer and Floating Point variables. Each type maintains its own namespace.

Note: Variables are part of the user environment stack. Thus a 'push' command creates a whole new set, copied forward from the previous level. If you 'pop' back, they are returned to their value in that level, and any new variables are lost.

See also: [!string]

Arguments

None- Displays the value of the String requested.

One or more – Sets the String to the concatenated value of the arguments and delimiters separating them if more than one.

Switches

/name= - Selects the name of the String variable to be other than the blank name. This can be any text value of any length.
/previous – Display only. Displays the value of the String variable as it was in the previous environment level after a 'push' command.
/clear – Setting only. Clears the String variable, setting it to the empty String: "".

Examples:

) string Hello World!
) write [!str]
Hello World!
) str/name=mystr1 My String One
) wr String Value: [!str/nam=mystr1]
String Value: My String One
)

SYSENV – Set/Display System Environment Values

Most Operating Systems support a set of system environment variables. Such as a 'Path'. These vary by Operating System. This command allows you to set or display these, but see the Operation section below.

Note: The variable name may be case insensitive or case sensitive, depending on Operating System.

Also see the [!SYSENV] Pseudomacro.

Operation

The System Environment values have no effect on the running of the CLI. They ARE however passed to any program you run using the EXECUTE (or XEQ) commands. Thus, changing these have no effect on the CLI, but will allow you to change settings for a program you will run from the CLI. You can also add new values.

These values are all kept together on the CLI stack, so you can PUSH, change them, and POP back to the original set.

Argument

If none – Displays all variables and their values.

If one argument is present, displays the value for that variable, or a message that none could be found. Note this message is not an exception, merely informational.

If two arguments are present, the first is the 'key' or name of the variable, and the second is the new value. It will add a kay/value pair if it does not exist.

Switches

/previous – Display the value(s) from the previous stack level. Cannot be used when setting a value.

Examples

) sysenv number_of_processors
number_of_processors = 4
) sysenv TEMP
temp = C:\Users\Dave_2\AppData\Local\Temp
) sysenv junk
System Environment Value Not Found for 'junk'.
)

SYSPROP – Display System Property Values

The CLI is written in the Java® programming language. Java maintains a set of system 'properties', and this pseudomacro allows you to retrieve them. You cannot alter them. Also see the [!SYSPROP] pseudomacro.

Note: The name may be case insensitive or case sensitive, depending on Operating System.

Argument

If none – Displays a list of all properties and values.
If one – The property name.

Switches

None.

Example

) sysprop java.vendor
Oracle Corporation
)

TIME – Display the current system time

Display the current system time using either the default format, or using one you provide via a switch.

Arguments

None.

Switches

/format = Specifies a format to use for displaying the time. This format must be in the same format as the Java java.text.SimpleDateFormat class definition. This overrides the CLI default format. See Section 4 on customization for a way to change the default time format.
/tz= Specifies a timezone ID to use. Must be from the set displayed using the /showalltimezones switch and is case sensitive. May also be from any timezone aliases specified in the resource property file.
/showalltimezones - Displays all available timezones. The output format is: '<timezone ID>' '<description>' : <sample time>.
Example output: 'CST' 'Central Standard Time' : 10:07:49

Note: You may also include date fields in your format.

Examples (note minimal uniqueness of switch in one):

) time
22:19:31
) time/format="hh:mm:ss aa"
10:19:31 PM

See also the [!time] pseudomacro which works much the same way.

TRACE – Set or display the 'Trace Mode'

The CLI can print out a trace of the command execution. It will print out the fully resolved text of any command executed, with a special prefix so you can see them.

If it is a simple command, the prefix will be '*** '. However, if the command is being run from within a macro, the prefix will be repeated '#' characters, with the number representing the call depth of the macro. So if your first macro calls a second one, then the commands in the second one will be prefixed with '## '. Also upon entering a macro, a line will be printed looking like: '# MACRO: <macro name>' .

Argument

'on', or 'off' – To turn tracing on or off.
If no argument – Display current setting.

TYPE – Displays the contents of a file

Simply displays the contents of a file or files to the user, with some optional display formats controlled by switches.

Arguments

Requires one or more filenames or templates. You can use a template to type multiple files. If no path is specified for a filename, the Searchlist will be searched for the file. The Searchlist is not searched for templates, only simple filenames. A template will just work in the current or specified directory.

Switches

/verbose – Display extra file info before the contents, such as the file's complete path.

/hex – Display the contents as bytes in hexadecimal.

Exceptions

If the file does not exist, a 'Class 2' exception message is returned.

VERSION – Display the current version of the CLI

Simply displays the version.

Arguments

None.

Switches

Only the usual /list=, 1=, 2=

Example

) ver
Version: 0.05
)

WRITE – Echoes text to the user or list file

This simply echoes the arguments back to the user after expansion of the arguments. Mostly useful in macros, or with the /list= switch to get text into a file.

Arguments

If there are no arguments, just a blank line is printed. Otherwise, there is no limit to the number of arguments.

Delimiter Note: The write command attempts to preserve the delimiter you use between arguments. These can be either spaces or commas. If you need custom spacing, simply put text you want displayed inside of double quotes.

Switches

The standard set: /list=, /1=, /2=

Note that the /list= switch (/l= for minimal uniqueness) is very useful on this command for pumping output to a file. See also the LISTFILE command.

Examples

See some of the pseudomacros for examples, such as [!date] .

Section 3 – Pseudomacros

Pseudomacros are much like inline 'functions' or 'methods' found in regular programming languages. They are an extensive set of built-in functions that return values of various kinds, and are evaluated before the CLI command is executed. All returned values are treated as text.

The syntax is: [!<name><optional switches> <optional arguments>]

And as usual these can be nested. So a pseudomacro can also be an argument to another pseudomacro. They can appear anywhere in a command line. They can even be in place of the command name itself, assuming they evaluate to a legal command name. They can also be used as values of switches.

Here is a simple example using text strings. First setting a string to a value using the STRING command, and then displaying it in some text using the WRITE command and the [!STRING] pseudomacro:

) string Slartibartfast
) write My name is not [!string]
My name is not Slartibartfast

As with commands, many of the pseudomacros accept switches. And as usual, the name of the pseudomacros and the switches are case insensitive, and can be minimally unique.

Conditional Control Pseudomacros

There is a special class of pseudomacros which allow conditional execution of your commands, either on a command line, or with multiple commands. This is explained in more detail in Section 1. They look like [!equal a b] (and many other comparisons), [!else], and [!end].

General Categories of Pseudomacros

- Control – Noted above because they are so unique in operation.
- Variables – String, Integer, and Floating point
- Math – Usual operators plus some functions like trigonometry, etc…
- System – Access to environment data like the current searchlist, environment stack level, etc…
- Utility – Things like displaying the date, time, etc…

Note: For ease of navigating this manual, each Pseudomacro will begin with a '!' in its title.

!BASENAMEPART – Returns the part of a filename(s) with no extension.

Returns just the name with no path or extension of the file or directory. This just parses the path or filename as a string, it does not have to exist. The 'Base name' is any part before the first period '.' found. If no '.' Is found, then the whole name is returned. An empty string is returned if there is no base part.

Arguments

One or more filenames, paths, or directory names.

Returns

The base names for each argument, separated by comma delimiters as needed, and an empty string if none exists.

Switches

None.

Examples

) wr Base name: [!basenamepart bin Test.dat [!path Test.java]]
Base name: bin,Test,Test
) wr Base name: [!bas TProps.java .dat timezones.txt]
Base name: TProps,,timezones
)

!CEILING – Returns a number rounded up the next integer.

The traditional math function to round up a floating point number.

Arguments

One floating point or integer value.

Returns

The next whole integer greater than or equal to the argument.

Switches

None.

!CLASS1 – Return the current 'Class 1' error handling action.

See Section 1 for more information on structured exception handling.

Arguments

None.

Returns

The text for the current Class 1 exception handling action.

Switches

/previous – Returns the setting from the previous environment stack level.

Example

) push
) wr Previous stack level Class 1 Setting: [!class1/prev]
Previous stack level Class 1 Setting: Error
) pop
) wr Previous stack level Class 1 Setting: [!class1/prev]
Error (10026) : No previous environment available on environment stack - Likely due to /previous switch.

!CLASS2 – Return the current 'Class 2' error handling action.

See Section 1 for more information on structured exception handling.

Arguments

None.

Returns

The text for the current Class 2 exception handling action.

Switches

/previous – Returns the setting from the previous environment stack level.

Example

) push
) wr Previous stack level Class 2 Setting: [!class2/prev]
Previous stack level Class 2 Setting: Warning
)

!CHARACTER – Returns the string made up of characters at the specified Unicode 'Code Points'

All characters, ASCII or not, can be represented numerically as part of the Unicode standard. If you need a character that is not easily typed, you can use this pseudomacro to generate it.

NOTE: To display international characters, your system, Operating System, console, etc.., must be configured for it. Most will not display beyond the ASCII range, which is the low end of the Unicode tables. You may often see a '?' in the event you are not configured for these character sets.

Arguments

One or more numeric values. These are converted to characters and concatenated to return a string.

You can use decimal, octal, or hex representations of the values. Examples for the ASCII letter 'A':

Decimal: 65 – Note: Decimal numbers MUST NOT start with a leading zero.
Octal: 0101 – Note: Octal numbers MUST start with a leading zero.
Hex: 0x41 – Note: Notice the 'x' after a '0'. Required for Hex.

Switches

None.

Example:

) write Test String: [!char 65 66 67]
Test String: ABC
)

!COS – Returns the Cosine of a number.

Returns the trigonometric function ‘Cosine’ for any number of degrees.

See also the [!degrees] pseudomacro which will convert radians to degrees.

Argument

One required – A number of degrees.

Switches

None.

!COUNT – Returns the number of arguments passed to it

Simply counts the number of arguments. Sounds trivial, but is useful when combined with other functions. See the example for something useful.

Arguments

Zero or more of any kind.

Switches

None.

Example

Get the number of text files in a directory.

) write The number of text files is: [!count [!filenames *.txt]]
The number of text files is: 4
)

Can also be used to test for the existence of a file, by giving [!filenames] a specific file name, and then testing for the return to be 0 or 1 .

!DATE – Return the current system date.

Returns the current system date using either the default format, or using one you provide via a switch.

No arguments.

Switches

/format = Specifies a format to use for displaying the date. This format must be in the same format as the Java java.text.SimpleDateFormat class definition. This overrides the CLI default format. See Section 4 on customization for a way to change the default date format.

/tz= Specifies a timezone ID to use. Must be from the set displayed using the TIME command's /showalltimezones switch and is case sensitive. May also be from any timezone aliases specified in the resource property file.

Note: You may also include time fields in your format.

Examples (note minimal uniqueness of switch in one):

) write Today is: [!date]
Today is: 09/05/15
) write Today is [!date/format="EEEE, MMMM d, yyyy"]
Today is Monday, September 7, 2015

!DEGREES – Converts Radians to Degrees.

Simply returns the number of degrees, given an arc angle in radians. Useful for the trigonometric functions that take degrees as an argument, such as [!cosine], [!sine], [!tangent], etc…

Also see [!pi] which returns the constant value for Pi, 3.1415926… etc…

Argument

One required – Angle in radians.

Switches

None.

Example

) write One pi Radians is [!degrees [!pi]] Degrees
One pi Radians is 180.0 Degrees
)

!DIRECTORY – Returns the user's current working directory.

Return the current working directory.

Argument

None.

Switches

/initial – Return the value it was when the CLI initially started up.
/previous – Return the value of the previous environment stack level. Will cause an error if there has not been a 'push' command to cause there to be a 'previous' level.

Note: You will get an error if you try and use both switches together, or with an argument.

Use Note: If you want to change to the 'initial' or 'previous' directory, use this pseudomacro as an argument to the 'DIRECTORY' command. For example, to change to the initial directory, do this:

) dir [!dir/i]

!ELSE – Begins an alternate conditional execution block.

This is a special pseudomacro that is for controlling conditional logic flow. Either within a line, or between commands. Specifically it is the optional alternate path part of if/then/else logic, that starts with a conditional test pseudomacro such as [!equal x y] and ends with a [!end].

See Section 1 under Conditional Execution for more detail.

Arguments

None.

Switches

None.

Example

) write A and B are [!equal A B]Equal[!else]Not Equal[!end]
A and B are Not Equal
) write 12 and 12 are [!iequal 12 12]Equal[!else]Not Equal[!end]
12 and 12 are Equal
)

!END – Ends a conditional execution block.

This is a special conditional execution control pseudomacro. It simply ends a conditional block of code, either in a command, or between commands.

See Section 1 under Conditional Execution for more detail.

Arguments

None.

Switches

None

Examples

See [!else] and conditional tests like [!equal] pseudomacro entries for examples.

!EQUAL – Conditional : Test for equality of two strings.

This begins a conditional code block. Specifically for comparison of two strings.

Arguments

Two strings required.

Switches

/ignorecase – Causes a case-insensitive comparison.

Example

Below is a series of lines that might be found in a macro. See Section 1 regarding macros. We are using the 'CREATE' command here with the /input switch to create the macro, and then we are running it.

) create/I calendar.cli
)) string/name=event1 09/20/15
)) [!equal [!date] [!string/name=event1]]
)) write You have an Event Scheduled Today!
)) [!else]
)) write No events today.
)) [!end]
)))
) calendar
No events today.
)

!EXPAND – Expand arguments by parsing tokens and delimiters

This will expand its arguments as if they had been typed in by the user on the command line.

Arguments

Zero or more arguments to be expanded.

Switches

/full – Performs a more complete parsing. See operation section below.
/argnum= - After expansion, full or not, only the specified text argument number will be returned (not delimiters). The number is 'zero relative'.

Operation

Sometimes in the CLI, particularly in String variables and date/time fields, multiple arguments and delimiters are put together in a single text field. They are not maintained as separate and distinct fields.

For example, doing:

) string a b c

Creates a single text field of 'a b c' with spaces. They will not be parsed again as separate text fields normally. So while you can do this:

) write Test (a b c)
Test a
Test b
Test c

Where the a, b, and c are treated as separate text fields, and the WRITE command sees them in a parenthesized group to be repeated. But in the following, see what happens:

) write Test ([!string])
Test a b c

This is because in the String variable, 'a b c' is a single text field. The same might occur if you used the [!read] pseudomacro to get user input.

To get around this, you use the [!expand] to re-parse a String or Strings as if they had originally been entered. Thus the solution to the above is:

) write Test ([!expand [!string]])
Test a
Test b
Test c

Another good use of this is if you put the result of a [!filenames] reference into a string.

/FULL switch explained:

It is possible to have special characters that the CLI normally sees as separate command parts. Like a pseudomacro reference itself. The basic operation of [!expand] will just treat those as text and not try and parse them down any further. However, with the /full switch, it will process them, and the resulting command line will behave exactly as if it had been entered by the user. Pseudomacro syntax in the text will evaluate, parentheses will cause repetition of commands, etc… Example:

) str "[!message 10025]"
) wr [!str]
[!message 10025]
) wr [!expand [!str]]
[!message 10025]
) wr [!expand/full [!str]]
Named String '%s' not found.

First we put the raw text of a psuedomacro into the String variable. We used double quotes around it to prevent execution. The CLI treats anything in double quotes as just plain text.

Then we expanded it normally, but the normal mode still just treated it as text.

Finally, with the /full switch, the expanded string was actually executed.

/argnum= Example

Say you want just the date a file was modified.
If 'Test.cli' was modified on : 2015-09-14 17:26:24 Then:

) wr Test.cli was changed on date: [!exp/arg=0 [!info/tlm Test.cli]]
Test.cli was changed on date: 2015-09-14

!EXTENSIONPART – Returns the extension part of a filename(s).

Just the extension after the first period '.' Is found in the filename. The file or directory does not have to exist. Returns an empty string if there is no extension.

Arguments

One or more filenames, paths, or directory names.

Returns

The extension for each argument, separated by comma delimiters as needed, and an empty string if none exists.

Switches

None.

Examples

) wr Base name: [!extensionpart Test.dat bin [!path Test.java] myfile.test.dat]
Base name: dat,,java,test.dat
)

!FABS – Returns the absolute value of a floating point number.

Simply returns the mathematical absolute value of a floating point number.

Argument

One numeric argument required.

Switches

None.

!FADD – Returns the result of adding two floating point numbers.

Simply adds two floating point numbers.

Arguments

Two numeric values required.

Switches

None.

!FDIVIDE – Floating Point division.

Simple floating point division.

Note: Division by zero will cause a Class 1 exception to occur. If the user has set the action to be 'warning' or 'ignore', then this yields the string "Infinity" rather than a numeric value.

Arguments

Two required. The first being the dividend, the second being the divisor.

Switches

None.

!FEQUAL – Conditional : Floating Point 'Equal to'

Test for Floating Point equality. One of the special conditional comparison pseudomacros. See Section 1 for more detail.

Note: Floating point numbers are very rarely 'equal'. They can be very close, bit may be off a very small bit. Better to multiply them by some power of 10 to get the accuracy you want, and then compare them as integers. A future release may allow an accuracy factor via a switch.

See the EQUAL Pseudomacro entry for more detail and examples of conditional logic.

Arguments

Two numeric arguments required.

Switches

None.

!FGE – Conditional : Floating Point 'Greater than or Equal to'

Test for Floating Point 'Greater than or Equal to'. One of the special conditional comparison pseudomacros. See Section 1 for more detail.

See the EQUAL Pseudomacro entry for more detail and examples of conditional logic.

Arguments

Two numeric arguments required.

Compares for : arg1 >= arg2

Switches

None.

!FGT – Conditional : Floating Point 'Greater Than'

Test for Floating Point 'Greater Than'. One of the special conditional comparison pseudomacros. See Section 1 for more detail.

See the EQUAL Pseudomacro entry for more detail and examples of conditional logic.

Arguments

Two numeric arguments required.

Compares for : arg1 > arg2

Switches

None.

!FILENAMES – Returns a list of files matching conditions

This operates much like the FILESTATUS command, except it only returns a comma separated list of names that match the conditions specified by the switches.

See FILESTATUS command for more detail.

Arguments

Zero or more file templates to match against. All templates are honored here, including use of the '#'. With no arguments, it lists files in the current working directory.

Switches

/sort= - Same fields can be sorted on as with the FILESTATUS command.
/pathname – Show full file pathname, not just the filename.
/directories – List only directories.
/nodirectories – Do not list directories.
/hidden – Show Operating System 'hidden' files. Default is to not show them.
/exclude= - Specifies a list of file templates to exclude from the operation. Accepts more than one if you put them in double quotes and separate them with the 'path separator' (Like ';' in Windows or ':' in Unix). Matches the filename only.

Filter Switches: Same set as in FILESTATUS command. Such as /tlmgt=2015-09-10, or /sizege=1000000 . See the FILESTATUS command for details.

Examples

) filestatus

Name	Type	Size	Modified
bin	**DIR**	**0**	**09/11/15 09:48:04**
docs	**DIR**	**0**	**09/11/15 09:48:11**
list.txt	**FILE**	**249**	**08/24/15 16:57:47**
props.txt	**FILE**	**4,473**	**08/31/15 16:15:25**
RegexFilter.class	**FILE**	**495**	**08/14/15 15:49:30**
RegexFilter.java	**FILE**	**315**	**08/14/15 15:49:26**
Test.java	**FILE**	**629**	**09/11/15 09:13:27**

) wr Files: [!filenames]
Files: bin,docs,list.txt,props.txt,RegexFilter.class,RegexFilter.java,Test.java
) wr Dirs: [!fi/dir]
Dirs: bin,docs
) wr Source: [!fi *.java]
Source: RegexFilter.java,Test.java
)

!FLE – Conditional : Floating Point 'Less than or Equal to'

Test for Floating Point 'Less than or Equal to'. One of the special conditional comparison pseudomacros. See Section 1 for more detail.

See the EQUAL Pseudomacro entry for more detail and examples of conditional logic.

Arguments

Two numeric arguments required.

Compares for : arg1 <= arg2

Switches

None.

!FLOOR – Returns a number rounded down to the next integer.

Rounds a number down to the next lower or equal integer. The opposite of [!CEILING].

Arguments

One floating point or integer value.

Returns

The next whole integer less than or equal to the argument.

Switches

None.

!FLT – Conditional : Floating Point 'Less Than'

Test for Floating Point 'Less Than'. One of the special conditional comparison pseudomacros. See Section 1 for more detail.

See the EQUAL Pseudomacro entry for more detail and examples of conditional logic.

Arguments

Two numeric arguments required.

Compares for : arg1 < arg2

Switches

None.

!FMULTIPLY – Floating Point Multiplication

Simply multiplies two floating point numbers.

Arguments

Two numeric values required.

Switches

None.

!FNAMEPART – Returns the filename part of a path.

Returns just the filename part of a full path. This just parses the path or filename as a string, it does not have to exist.

Arguments

One or more filenames, paths, or directory names.

Returns

The file name for each argument, separated by comma delimiters as needed

Switches

None.

Examples

) wr Filename: [!fnamepart C:\users\westly\docs\resume.doc]
Filename: resume.doc
)

!FNEQUAL – Conditional : Floating Point 'Not Equal to'

Test for Floating Point 'Not Equal to'. One of the special conditional comparison pseudomacros. See Section 1 for more detail.

See the EQUAL Pseudomacro entry for more detail and examples of conditional logic.

Arguments

Two numeric arguments required.

Switches

None.

!FSUBTRACT – Floating Point Subtraction

Simply subtracts one Floating Point number from another.

Arguments

Two numeric values required.

Arg 1 – The number to subtract from. Aka the 'Minuend'.
Arg 2 – The number to be subtracted from Arg 1. Aka the 'Subtrahend'.

Switches

None.

!FVAR – Returns the value of a Floating Point Variable

This returns the value of a 'Floating Point' variable. There is a default unnamed variable, or you can name your variables. You use the FVAR command to set the variable value. The unnamed variable will have a value of 0.0 if not set. If you access a named variable and it has not been set, you will get an error back.

Arguments

None.

Switches

/name= - Optional. Specifies the name of the variable. Uses the unnamed variable if not specified. You can have as many of these as your system memory allows. Note these names are distinct from other variable types, so a name of 'myvar' for an FVAR is not the same variable as 'myvar' for a STRING variable.

/previous – Returns the value from the previous level of the user environment stack. See PUSH and POP commands, as well as Section 1.

Example

) fvar/name=var1 123
) fvar/name=var2 234
) write Sum of var1 and var2: [!fadd [!fvar/name=var1] [!fvar/name=var2]]
Sum of var1 and var2: 357.0

Note the use of the [!fadd] Floating Point addition pseudomacro.

!IABS – Returns the 'Absolute Value' of an integer

Simply returns the mathematical absolute value of an integer number.

Argument

One numeric argument required.

Note: A floating point argument will be truncated (floor) to an integer before processing.

Switches

None.

!IADD – Integer Addition

Simply adds two integer numbers.

Arguments

Two numeric values required.

Note: A floating point argument will be truncated (floor) to an integer before processing.

Switches

None.

!IDIVIDE – Integer Division

Simple integer division.

Note: Division by zero will cause a Class 1 exception to occur. If the user has set the action to be 'warning' or 'ignore', then this yields the string "Infinity" rather than a numeric value.

Arguments

Two numbers required. The first being the dividend, the second being the divisor.

Note: A floating point argument will be truncated (floor) to an integer before processing.

Switches

None.

!IEQUAL – Conditional : Integer 'Equal To'

Test for Integer equality. One of the special conditional comparison pseudomacros. See Section 1 for more detail.

See the EQUAL Pseudomacro entry for more detail and examples of conditional logic.

Arguments

Two integer arguments required.

Note: A floating point argument will cause a Class 1 exception. Use [!floor] on the argument if you need to compare as integers, or use [!fequal] which will accept integers as arguments.

Switches

None.

!IGE – Conditional : Integer 'Greater Than or Equal To'

Test for Integer 'Greater than or Equal to'. One of the special conditional comparison pseudomacros. See Section 1 for more detail.

See the EQUAL Pseudomacro entry for more detail and examples of conditional logic.

Arguments

Two integer arguments required.

Compares for : arg1 >= arg2

Note: A floating point argument will cause a Class 1 exception. Use [!floor] on the argument if you need to compare as integers, or use [!fge] which will accept integers as arguments.

Switches

None.

!IGT – Conditional : Integer 'Greater Than'

Test for Integer 'Greater Than. One of the special conditional comparison pseudomacros. See Section 1 for more detail.

See the EQUAL Pseudomacro entry for more detail and examples of conditional logic.

Arguments

Two integer arguments required.

Compares for : arg1 > arg2

Note: A floating point argument will cause a Class 1 exception. Use [!floor] on the argument if you need to compare as integers, or use [!fgt] which will accept integers as arguments.

Switches

None.

!ILE – Conditional : Integer 'Less Than or Equal To'

Test for Integer 'Less than or Equal to'. One of the special conditional comparison pseudomacros. See Section 1 for more detail.

See the EQUAL Pseudomacro entry for more detail and examples of conditional logic.

Arguments

Two integer arguments required.

Compares for : arg1 <= arg2

Note: A floating point argument will cause a Class 1 exception. Use [!floor] on the argument if you need to compare as integers, or use [!fle] which will accept integers as arguments.

Switches

None.

!ILT – Conditional : Integer 'Less Than'

Test for Integer 'Less Than'. One of the special conditional comparison pseudomacros. See Section 1 for more detail.

See the EQUAL Pseudomacro entry for more detail and examples of conditional logic.

Arguments

Two integer arguments required.

Compares for : arg1 < arg2

Note: A floating point argument will cause a Class 1 exception. Use [!floor] on the argument if you need to compare as integers, or use [!flt] which will accept integers as arguments.

Switches

None.

!IMOD – Integer Modulo Function

Returns the 'MOD' function of two integers. This is also sometimes called the 'remainder' function.

Arguments

Two numbers required. Any floating point numbers will be truncated (floor) to integers before the operation.

Returns 'arg1 mod arg2'. Sometimes written in languages as 'arg1 % arg2'.

Switches

None.

!IMULTIPLY – Integer Multiplication

Simply multiplies two integer numbers.

Arguments

Two numeric values required.

Note: A floating point argument will be truncated (floor) to an integer before processing.

Switches

None.

!INEQUAL – Conditional : Integer 'Not Equal To'

Test for Integer inequality. One of the special conditional comparison pseudomacros. See Section 1 for more detail.

See the EQUAL Pseudomacro entry for more detail and examples of conditional logic.

Arguments

Two integer arguments required.

Note: A floating point argument will cause a Class 1 exception. Use [!floor] on the argument if you need to compare as integers, or use [!fequal] which will accept integers as arguments.

Switches

None.

!ISUBTRACT – Integer Subtraction

Simply subtracts two integer numbers.

Arguments

Two numeric values required.

Arg 1 – The number to subtract from. Aka the 'Minuend'.
Arg 2 – The number to be subtracted from Arg 1. Aka the 'Subtrahend'.

Note: A floating point argument will be truncated (floor) to an integer before processing.

Switches

None.

!INFO – Returns various file information fields for a file or directory

This allows you to access a specific field of information about a file or directory. For example just its size in bytes.

Argument

One name or pathname to a file or directory. It must exist or a Class 1 error will be raised.

Switches

If none – Just the filename itself is returned.
/size – The file size in bytes is returned.
/type – The file type, such as 'FILE', or 'DIR' etc… is returned.
/tcr – The time of creation.
/tla – The time last accessed.
/tlm – The time last modified

NOTE: For the time switches, the format used is the default date/time as defined in the CLI's resource property file. See Section #### for more information. It is also two fields by default, a date and a time, returned as a single string with a space between them. So if you want to break them apart, use the [!expand] pseudomacro using the /argnum= switch.

Example

) write The Test macro is [!info/siz Test.cli] bytes in size.
The Test macro is 174 bytes in size.
) write The Test macro was last modified at [!info/tlm Test.cli] .
The Test macro was last modified at 2015-09-14 17:26:24 .
)

!IVAR – Returns the value of an Integer Variable

This returns the value of an Integer variable. There is a default unnamed variable, or you can name your variables. You use the IVAR command to set the variable value. The unnamed variable will have a value of 0 if not set. If you access a named variable and it has not been set, you will get an error back.

Arguments

None.

Switches

/name= - Optional. Specifies the name of the variable. Uses the unnamed variable if not specified. You can have as many of these as your system memory allows. Note these names are distinct from other variable types, so a name of 'myvar' for an IVAR is not the same variable as 'myvar' for an FVAR variable.

/previous – Returns the value from the previous level of the user environment stack. See PUSH and POP commands, as well as Section 1.

Example

) ivar/name=var1 123
) ivar/name=var2 234
) write Sum of var1 and var2: [!iadd [!ivar/name=var1] [!ivar/name=var2]]
Sum of var1 and var2: 357

Note the use of the [!iadd] Integer addition pseudomacro.

!LASTERRORCODE – Returns the error code for the last command.

Returns the integer error code associated with the last command executed. Zero '0' for no error.

See also: [!lastexceptiontype]

Arguments

None.

Switches

None.

Example:

) write Hello World!
Hellow World!
) wr Last Error was: [!lasterr]
Last Error was: 0
) wr String is: [!str foo]
Error (10024) : Pseudomacro does not accept arguments.
) wr Last Error was: [!lasterr]
Last Error was: 10024
)

!LASTEXCEPTIONTYPE – Returns the exception for the last command.

Returns the exception type associated with the last command executed. 'None' if no error had been reported.

Note: These correspond to the 'Class 1' and 'Class 2' exception levels. See the discussion of error handling in Section 1.

See also: [!lasterrorcode]

Arguments

None.

Switches

None.

Example:

) write Hello World!
Hellow World!
) wr Last Exception was: [!lastex]
Last Exception was: None
) wr String is: [!str foo]
Error (10024) : Pseudomacro does not accept arguments.
) wr Last Exception was: [!lastex]
Last Exception was: Error
)

!LEVEL – Returns the current 'Environment Stack' level

Returns the integer number of levels on the environment stack. If you have not done any 'push' commands, you are at the first level '1'.

Arguments

None.

Switches

None.

Example:

) wr The current level is: [!lev]
The current level is: 1
)

!LISTFILE – Returns the value of the LISTFILE Environment Setting

The LISTFILE user environment variable exists on the environment stack. This pseudomacro returns the value. See the LISTFILE command and Section 1 for more information on this variable.

Argument

None.

Switches

/initial – Returns the initial setting. Usually 'System.out' or 'stdout', which are synonymous.
/previous – Returns the previous setting on the environment stack. No argument allowed. Causes an error if there is no previous level on the stack (I.E. no 'PUSH' command has happened)

Example

) write Current List File is: [!listfile]
Current List File is: System.out
) write Current List File is: [!list/prev]
Error (10026) : No previous environment available on environment stack - Likely due to /previous switch.
)

!LOGFILE – Return the current CLI logging destination

The LOGFILE user environment variable exists on the environment stack. This pseudomacro allows you to get the current setting.

Argument

None.

Switches

/previous – Returns the previous setting on the environment stack. No argument allowed. Causes an error if there is no previous level on the stack (I.E. no 'PUSH' command has happened)

Operation

If logging is disabled, this will return "null", rather than a filename. Also see the LOGFILE command which behaves a little differently.

!LOGN – Returns the 'Natural Log' of a Floating Point number.

Returns the math function 'Natural Log' for a number. Often expressed as Log_n of x.

Argument

One required – A number you want the Log_n of.

Switches

None.

!LOG10 – Returns the ‘ Log Base 10’ of a Floating Point number.

Returns the math function ‘Log Base 10’ for a number. Often expressed as Log_{10} of x.

Argument

One required – A number you want the Log_{10} of.

Switches

None.

!MESSAGE – Returns the text for a message key

Similar to the MESSAGE command, this returns the text associated with a key, such as from an error code.

Argument

One and only one required. They message key.

Example:

) write The text for error code 10025 is: [!message 10025]
The text for error code 10025 is: Named String '%s' not found.
)

!NEQUAL – Conditional : String 'Not Equal To'

This begins a conditional code block. Specifically for inequality of two strings.

Note: This is synonymous with [!SNEQUAL] (String Not Equal).

Arguments

Two strings required.

Switches

/ignorecase – Causes a case-insensitive comparison.

Example

See [!EQUAL] for an example of conditional execution.

!PARENTPART – Returns the parent part of a pathname.

Returns just the path of the directory level above the referenced file or directory. This just parses the path as a string, so the file or directory does not actually have to exist.

Arguments

One or more filenames, paths, or directory names.

Returns

The parent path for each argument, separated by comma delimiters as needed, or an empty string if none exists (you are at the root).

Switches

None.

Examples

) wr Parent: [!parentpart C:\users\inigo\docs\resume.doc]
Parent: C:\users\inigo\docs
)

!PATHNAME – Returns the resolved full path to a file or directory

This simply returns the full pathname of the file or directory, or various parts of the name. It will find it on the searchlist if just a name is given. It will return an error if the file does not exist. See also the PATHNAME command.

Argument

Only one allowed, which is the filename of the file or directory you want the path info for.

Switches

If none – Returns the full 'Canonical Path' to the file or directory. Or returns a 'does not exist' error.

The next switches are mutually exclusive. You can only choose one.

/filename – Returns just the filename part of a path. The file or directory does NOT need to exist.

/extension – Returns just the 'extension' of the filename, such as 'txt' for a file named 'billing.txt'. The file does NOT need to exist.

/parent – Returns the parent directory of the specified file or directory. If a full path is specified as an argument, then the file does not need to exist. However if just a simple file or directory is specified, it will try and resolve its location first, and then return the parent, so you could then get a 'does not exist' error. This switch is thus useful for finding the directory containing a file that is found someplace on the searchlist.

!PI – Returns the constant value for Pi (π)

Simply provides the constant value for Pi (π) : 3.1415926…. etc…

Arguments

None.

Switches

None.

Example

) wr Pi is: [!pi]
Pi is: 3.141592653589793
)

!PID – Return the CLI's 'Process ID'

This returns the integer Process ID as assigned by the Operating System.

Note: This can be useful for things like creating temporary working files that should not conflict with other programs, as the PID is unique for any given executing instance on known Operating Systems.

See also the PID Command.

Arguments

None.

Switches

None .

Example

) wr The PID for this CLI is: [!pid]
The PID for this CLI is: 7234
) create TempFile_[!pid]
)

!RANDOM – Returns a random number x where: 0.0 <= x < 1.0

Just returns a random floating point number with an approximately even distribution across the range greater than or equal to 0.0 and less than 1.0 .

Arguments

None.

Switches

None.

!READ – Prompt the user and read input

This will use any arguments as a prompt to the user, and then wait for the user to enter text. The text is expanded as if it was typed into the command line.

Note: Any special characters such as angle brackets, parenthesis, and other pseudomacro references will NOT be further expanded.

Arguments

The arguments are used as text to prompt the user. One space is added at the end. Delimiters are preserved.

Switches

None.

Example:

) string [!read Please enter your name:]
Please enter your name: Trillian
) write Your name is [!str]
Your name is Trillian
)

The above example took the text entered and set the default String variable to it. Then the 'write' command used the [!string] pseudomacro to display the value of the String.

!ROOTPART – Returns the filesystem root part of a pathname.

Returns just the root of the file or directory path. This just parses the path as a string, so the file or directory does not actually have to exist. Note that if a full path is not specified, a root may not be found. The root will vary by Operating System. For example, in Windows it might be “C:\” or “F:\” etc…

Arguments

One or more filenames, paths, or directory names.

Returns

The file system root for each argument, separated by comma delimiters as needed, or an empty string if none exists (you are at the root).

Switches

None.

Examples

) wr Root: [!rootpart C:\users\inigo\docs\resume.doc]
Root: C:
)

!ROUND – Rounds a Floating Point number to the nearest integer

Returns the usual mathematical function for rounding floating point numbers to integers. That is, for values less than one half a number, it rounds down to the full integer, and for numbers one half and above it rounds up to the next highest integer. Thus 1.49999 rounds to 1 and 1.5 rounds to 2.

Argument

One numeric value.

Switches

None.

!SEARCHLIST – Returns the current value of the SEARCHLIST

Just returns the current (or previous stack level with /previous switch) value of the SEARCHLIST user environment variable.

See also the SEARCHLIST command.

Arguments

None.

Switches

/previous – Returns the value from the previous level of the user environment stack. Causes a Class 1 exception if there is no previous level (no previous PUSH). If Class 1 is set to Ignore or Warning, returns an empty searchlist.

!SEQUAL – Conditional : Alternate form of !EQUAL

This begins a conditional code block. Specifically for comparison of equality for two strings.

It is a synonym for [!EQUAL], but starts with an 'S' to be consistent with the naming of other conditional pseudomacros that start with the letter of their type (S, I, F).

Arguments

Two strings required.

Switches

/ignorecase – Causes a case-insensitive comparison.

Example

Below is a series of lines that might be found in a macro. See Section 1 regarding macros. We are using the 'CREATE' command here with the /input switch to create the macro, and then we are running it.

) create/I calendar.cli
)) string/name=event1 09/20/15
)) [!sequal [!date] [!string/name=event1]]
)) write You have an Event Scheduled Today!
)) [!else]
)) write No events today.
)) [!end]
)))
) calendar
No events today.
)

!SGE – Conditional : String 'Greater Than or Equal To'

This begins a conditional code block. Specifically for comparison of two strings.

See Section 1 on conditional execution.

Arguments

Two strings required. Executes the block if Arg1 is lexically >= Arg2.

Switches

/ignorecase – Causes a case-insensitive comparison.

!SGT – Conditional : String 'Greater Than'

This begins a conditional code block. Specifically for comparison of two strings.

See Section 1 on conditional execution.

Arguments

Two strings required. Executes the block if Arg1 is lexically > Arg2.

Switches

/ignorecase – Causes a case-insensitive comparison.

!SIN – Return the Sine function of a Floating Point number

Returns the trigonometric function 'Sine' for any number of degrees.

See also the [!degrees] pseudomacro which will convert radians to degrees.

Argument

One required – A number of degrees.

Switches

None.

!SLE – Conditional : String 'Less Than or Equal To'

This begins a conditional code block. Specifically for comparison of two strings.

See Section 1 on conditional execution.

Arguments

Two strings required. Executes the block if Arg1 is lexically <= Arg2.

Switches

/ignorecase – Causes a case-insensitive comparison.

!SLT – Conditional : String 'Less Than'

This begins a conditional code block. Specifically for comparison of two strings.

See Section 1 on conditional execution.

Arguments

Two strings required. Executes the block if Arg1 is lexically < Arg2.

Switches

/ignorecase – Causes a case-insensitive comparison.

!SNEQUAL – Conditional : Alternate form of [!NEQUAL]

This begins a conditional code block. Specifically for inequality of two strings.

Note: This is synonymous with [!NEQUAL] (Not Equal).

Arguments

Two strings required.

Switches

/ignorecase – Causes a case-insensitive comparison.

Example

See [!EQUAL] for an example of conditional execution.

!STRING – Returns a String variable or length. With optional functions.

This returns the value of a String variable. There is a default unnamed string, or you can name your Strings. You use the STRING command to set the variable value. The unnamed String variable will have a value of an empty string. I.E. "" of length 0. If you access a named String variable and it has not been set, you will get an error back.

There are also String operations possible with some switches. You can get the length of the string, take a substring of it, and find the position of another string within the variable.

Note: String offsets are zero relative. That is, the first character is at position 0.

Example:

) string This is the un-named string
) string/name=mystr This is my String
) write The value of the unnamed string is: [!str]
The value of the unnamed string is: This is the un-named string
) write The value of my named string is: [!str/na=mystr]
The value of my named string is: This is my String

Switches

/name= - Specifies the name of the String variable. You can have as many of these as your system memory allows.
/previous – Returns the value from the previous level of the user environment stack. See PUSH and POP commands, as well as Section 1.
/length – Returns the length of the string in characters.
/index – Requires an argument in the Pseudomacro and returns the position (zero relative) of the first occurrence of the argument text within this string. Or '-1' if it is not found.
/start= - Specifies the starting position of a substring of the string variable to return. The default if not present is 0.
/end= - Specifies the ending position of a substring of the string variable to return. The default if not present is the end of the string. NOTE: The end position of the string would be its length-1. So "abc" would be indexed with "a" at position 0, "b" at position 1, and "c" at position 2.

Argument

One argument is allowed only with the /index switch. In which case the argument text is searched for in the String variable.

Examples

Note: Minimal uniqueness used for switches and commands etc…

) string abcdefg
) wri Substring: [!str/start=2/end=4]
Substring: cde
) wri Substring: [!str/end=4]
Substring: abcde
) wr Length: [!str/len]
Length: 7
) wr Index of efg: [!str/index efg]
Index of efg: 4
) wr Index of xyz: [!str/index xyz]
Index of xyz: -1

!SYSENV – Return System Environment Values

Most Operating Systems support a set of system environment variables. Such as a 'Path'. These vary by Operating System. This command allows display of these, but they cannot be modified from here.

Note: The variable name may be case insensitive or case sensitive, depending on Operating System.

Argument

One – The variable name.

Switches

/previous – Display the value from the previous stack level.

Example

) write Your Operating System Is : [!sysenv OS]
Your Operating System Is : Windows_NT
) write The value for the 'junk' setting is: [!sysenv junk]
Error (10069) : System Environment Value for 'junk' not found.

Note: Unlike the SYSENV command, the pseudomacro will throw a Class 1 exception if the key is not found.

!SYSPROP – Return System Property Values

The CLI is written in the Java programming language. Java maintains a set of system 'properties', and this pseudomacro allows you to retrieve them. You cannot alter them. Also see the SYSPROP command.

Note: The name may be case insensitive or case sensitive, depending on Operating System.

Argument

One – The property name.

Switches

None.

Example

) write Your Java Vendor Is : [!sysprop java.vendor]
Your Java Vendor Is : Oracle Corporation
)

!TAN – Return the Tangent function of a Floating Point number

Returns the trigonometric function 'Tangent' for any number of degrees.

See also the [!degrees] pseudomacro which will convert radians to degrees.

Argument

One required – A number of degrees.

Switches

None.

!TIME – Return the current system Time of day

Returns the current system time using either the default format, or using one you provide via a switch.

No arguments.

Switches

/format = Specifies a format to use for displaying the time. This format must be in the same format as the Java java.text.SimpleDateFormat class definition. This overrides the CLI default format. See Section 4 on customization for a way to change the default time format.

/tz= Specifies a timezone ID to use. Must be from the set displayed using the TIME command's /showalltimezones switch and is case sensitive. May also be from any timezone aliases specified in the resource property file.

Note: You may also include date fields in your format.

Examples (note minimal uniqueness of switch in one):

) write The current time is: [!time]
The current time is: 22:23:12
) write The time is now: [!time/format="hh:mm:ss aa"]
The time is now: 10:24:31 PM

!VERSION – Returns the current version of the CLI

Simply displays the version.

Arguments

None.

Switches

None.

Example

) wr Current CLI Rev is: [!ver]
Current CLI Rev is: 0.05

Section 4 – Startup Customization

When the CLI starts up, there are two ways you can have it initialize with some values other than the defaults. One is with a 'startup macro', and the other is by providing your own 'CLI.properties' file. Or both.

NOTE: These methods are only available in the paid package levels. Specifically, the 'startup macro' is not available in the 'Free' version. Only in the base paid level on up. The 'CLI.properties' method is only available in the 'Commercial' version on up.

Startup Macro

Specify the name of the macro on the startup line. Either as the only argument, or as a value to the '-i' switch (as in Input).

Example using Windows, where the startup macro is a file named 'startup.cli':

> java –jar CLI.jar startup.cli

Or

> java –jar CLI.jar –i startup.cli

If the macro ends with the BYE command, the CLI will exit after execution. Otherwise, it will run the macro, and then just drop into the usual user prompt.

ANY commands can be run in this macro. However it is not currently possible to pass any arguments to the macro at this time.

Useful things to do in the macro might be:

DIRECTORY command – Change your current working directory.
SEARCHLIST command – Set a searchlist.
PROMPT command – Set the CLI prompt.
PREFIX command – Set the CLI prefix.

Or any other environment settings.

Or you can just use this capability to run CLI macros as batch jobs.

CLI.properties

The other startup customization can be done by specifying your own property file at startup. There is a default in the JAR file, but you can override it.

NOTE: be very careful with this, as you can break things! The default file is provided if you purchase the Commercial level package or above. Also note that the default may change from version to version.

Example file:

```
# Property file used for various parameters for the CLI program
#
ResourceFile=CLI_Resources.properties
Commands=CLI_Commands.properties
Pseudomacros=CLI_Pseudomacros.properties
searchlist=C:\\Users\\Dave_2\\Documents\\Dev
directory=.
cmdSeparator=;
# Regular Expression (Regex) for input of date/time.  As with FILESTATUS time filters.
DateTimeInputRegex=([0-9]{4})-([0-9]{1,2})-([0-9]{1,2})(:([0-9]{1,2}))?(:([0-9]{1,2}))?(:([0-
9]{1,2}))?
# The fields represented by the regex groups in the above regex.
DateTimeInputFields=y+M+d+H+m+s
# Default data fields to display in the FILESTATUS command.
filestatus_fields=name+type+size+tlm
#
# End
#
```

You specify your alternate version as startup using the '-env' switch as follows:

> java –jar CLI.jar –env startup.properties

Value definitions (case matters!):

ResourceFile – The name of a 'resource bundle file'. See Section 5. This can only be changed in the Enterprise package level.
Commands – The name of a command definition property file. See Section 5. This can only be changed in the Commercial package level.
Pseudomacros – The name of a pseudomacro definition file. See Section 5. This can only be changed in the Commercial package level.
searchlist – The SEARCHLIST value to use at startup.
directory – The current working directory at startup. Note '.' Means whatever your CWD is when the java runtime is started.
cmdSeparator – The character to use for separating multiple commands on a command line.

DateTimeInputRegex and **DateTimeInputFields** – These together specify the format to use when inputting date and time stamps for commands that accept them, such as the FILESTATUS command's date/time range filters. A knowledge of Regular Expressions is helpful here, but simply, the 'regex groups' in parens correspond with the fields separated by plus signs in the field list. Those fields, such as 'M', correspond to the Java language formatting fields. Thus: 'y' – year, 'M' month, 'd' – day of month, 'H' Hour of day 0-23, 'm' – minute of hour, 's' – second of minute. In the above file for example, the first regex field '([0-9]{4})' means "a group consisting of 4 characters in the range '0' thru '9'" (4 digits), and it corresponds to the 'y' from the field definition. Thus the entry must start with the form 'yyyy' or a 4 digit year.

So the above will recognize: 2015-09-13:11:00:00 – For 11 AM on September 13th, 2015. Note the hyphens and colons also must match, as hyphens and colons.

The '?' symbols mean that groups are optional. This allows, in the above default, for you to leave off the time of day fields. They would be set to zero if not entered by the user. As defined here, either the entire set of time of day groups can be left off, or an individual group, in which case if you had only 2 time groups entered, they would be hour and minute (seconds would be assumed to be zero). With one time group entered, it would be assumed to be hours.

Experimentation is likely needed.

Section 5 – Advanced Customization

In the Commercial and Enterprise level packages, you can extend the CLI by adding your own commands and pseudomacros! You can also change the text used by the CLI, for example you could change the names of commands and pseudomacros, or error messages or other displayed text. You might want to do this just for your own preference, or for internationalization specific to a language. There is virtually no hard-coded text in the CLI.

As mentioned back in Section 1, the CLI is written in Java®. As such, it has some powerful capabilities, such as being able to dynamically load compiled code at runtime. This allows all of the code for commands and pseudomacros to be loaded based on parameters. Each command or pseudomacro is represented by a Java 'class'.

If you purchase the Commercial or Enterprise package levels of the CLI, you will be provided with some base Java code and examples that you can 'extend' to create your own commands or pseudomacros. You will also get the raw binaries of other code so you can build your own 'JAR' file.

To override the defaults built into the delivered JAR file, you use three of the fields found in the 'CLI.properties' file discussed in Section 4. Specifically: 'ResourceFile', 'Commands', and 'Pseudomacros'.

Index

D

E

F

G

H

I

J

L

M

N

P

R

S

T

U

V

W

X

Π