

MBARI Carbon WG Console Help

Command syntax help is organized using the following topics:

- REG built-in register variables
- QUEUE sample queue
- FSYS file system
- DIAG diagnostics/debug
- GET get configuration/state/IO
- SET set configuration/state/IO
- CLR clear IO
- PULSE momentary IO cycling
- LI licor control
- SCR script control
- MAC macros

Group [REG]

CREG

```
use : creg [<reg>] [op <val>]
brief : show, manipulate counter reg (CX0:SCR_CX_COUNT)
input : reg - register [CXn[*]]
input : op - operator [=,--,++,+,-,*,/,%,]
input : val - value [int32]
return : requested value(s)
```

DREG

```
use : dreg [<reg>] [op <val>]
brief : show, manipulate double reg (DX0:SCR_DX_COUNT)
input : reg - register [DXn[*]]
input : op - operator [=,--,++,+,-,*,/,%,]
input : val - value [double]
return : requested value(s)
```

FREG

```
use : freg [<reg>] [op [<val>]]
brief : show, manipulate flag(s) (0:31)
input : reg - register [FXn[*]]
input : op - operator [=,!]
input : val - value [1|0]
return : requested value(s)
```

Group [QUEUE]

QCOUNT

```
use : qcount
brief : return queue entries
return : number of queue entries
```

QCLEAR

```
use : qclear
brief : delete all queue entries
return : none
```

QPEEK

```
use : qpeek [<first>] [<n>]
brief : examine queue entries (don't delete)
input : first - starting queue entry
```

```
input : n      - number of entries or '*'
return : decoded queue entries
```

QPOP

```
use      : qpop [Q] <n>
brief    : delete and optionally show next (oldest) n entries from queue
input    : n - number of entries or '*' for all
input    : Q - suppress output (just delete)
return   : oldest n entries in the queue (or none if Q specified)
```

QSHOW

```
use      : qshow
brief    : print queue summary
return   : queue summary
```

Group [FSYS]

CAT

```
use      : cat <file>
brief    : show file contents
input    : file - file to show
return   : file contents
```

CFGEX

```
use      : cfgex <file>
brief    : export current configuration to file
input    : file - destination file
          [NOTE : file will be overwritten if it exists]
return   : none
```

CP|MV

```
use      : cp <src> <dest>
brief    : copy contents from src to dest
input    : src - file to copy FROM
input    : dest - file to copy TO
return   : none
```

LC

```
use      : lc <file>
brief    : count lines in file
          [any console input interrupts]
input    : file - file to evaluate
return   : number of lines, execution time
```

LESS

```
use      : less <file> [<lines>]
brief    : page a text file to the console (with search)
input    : file - file to evaluate
input    : lines - number of text lines per page
return   : none
notes    :
```

```
Command Summary
Move      : Up [u U]   Down [d D CR]
Jump      : First [g <] Last [G >]
Search    : Find [f /] Next [n N]
Session   : Help [h H ?] Quit [q Q x X]
```

LOG

```
use      : log <msg>
brief    : add log annotation
input    : msg - message to log
return   : none
```

LS|DIR

```
use      : ls [<dir>]
brief    : list file info
input    : dir - directory [default: root, i.e. 0:/]
return   : list of files in directory
```

LSEG

```
use      : lseg <hnd> [<val>]
brief    : get/set log segment size
input    : hnd - log handle 0:syslog 1:data 2:raw
input    : val - log segment size, n>0
           may also use K,M,G suffix (e.g. 10M) for KByte, MByte, GByte
return   : none
```

LSHOW

```
use      : lshow [<hnd>]
brief    : show log summary
input    : hnd - log handle 0:syslog 1:data 2:raw
return   : none
```

REN

```
use      : ren <from> <to>
brief    : change file name
input    : from - current file name
input    : to   - new file name
return   : none
```

RM

```
use      : rm <file> [<file>...]
brief    : delete one or more files
input    : file - file to delete
return   : none
```

STATS

```
use      : stats <vol>
brief    : show file system information
input    : vol - volume
return   : file system summary
```

UPLOAD

```
use      : upload <dest>
brief    : upload ASCII file to root directory
input    : dest - file name
           dest will be overwritten if it exists
           otherwise, it will be created
           upload times out after 4 s, or may be terminated using CTRL_D
           [NOTE : not a production feature;
           some terminal emulators may not support direct ASCII upload]
return   : none
```

Group [DIAG]

HELP[?]

```
use      : help [V]
brief    : show help info
input    : V - verbose output (this file) (optional)
return   : help information
```

HIST[!]

```
use      : hist [v] [<id>]
brief    : show command history, execute previous command
input    : v   - verbose output (optional, ignored if id specified)
input    : <id> - command ID (integer) (optional)
return   : command history by default, executes queued command if specified
note     : up/down arrow keys may be used to traverse the queue
```

CON

```
use      : con <port>
brief    : open console to device
           [terminate session using CTRL_D]
input    : port - serial port [LI|HO]
return   : none
```

DEBUG

```
use      : debug [<op> <mask>]
brief    : show/set per-module debug output flags
           MOD_ACT =0x0001 // cli_impl
           MOD_TMR =0x0002 // timers
           MOD_SCR =0x0004 // script
           MOD_LIC =0x0008 // licor
           MOD_SER =0x0010 // ioserial
           MOD_CFG =0x0020 // config
           MOD_MAIN=0x1000 // main
           MOD_TOKS=0x2000 // command token

input    : op - operator [+:set -:clear =:assign]
input    : mask - hex bit field [0-FFFFFFFF]
return   : debug flags
```

PMEM

```
use      : pmem
brief    : show memory pool contents
return   : memory pool contents
```

ECHO

```
use      : echo [<op><dest>] [<msg>...]
brief    : print messages using variable expansion escape character support
input    : op - operator [+:enable -:disable]
input    : dest - destination [C:console H:host]
input    : msg - text, variables, registers
           quoted strings preserve space, unquoted strings are space-delimited
           variables and registers are denoted by $ (e.g. $verbose, $CX2)
           escape characters are supported in quoted strings [\r\n\t\v\xDD]

return   : expanded message
```

VER

```
use      : ver
brief    : show firmware version
return   : version information
```

UPTIME

```
use      : uptime
brief    : return elapsed time since last power cycle
return   : time since last power cycle
```

RESET

```
use      : reboot
brief    : reboot the firmware
return   : none
```

HELLO

```
use      : hello [<msg>...]
brief    : print test messages to console
input    : msg - text, variables, registers
           quoted strings preserve space, unquoted strings are space-delimited
           variables and registers are denoted by $ (e.g. $verbose, $CX2)
           escape characters are supported in quoted strings [\r\n\t\v\xDD]

return   : 'hello' [args...]
```

SWEEP

```
use      : sweep <delay_ms>
brief    : turn all IO bits on and then off in order
input    : delay_ms - switching delay
return   : sets all IO bits LO (except serial transceivers)
```

PROTO

```
use      : proto (args)
brief    : runs selected prototype code [experts only]
          : enabled code depends on compile-time configuration in cwgsys.h
input    : args - arguments (varies)
return   : varies
```

Group [GET]

GET AD

```
use      : get ad <port> [DXn]
brief    : show A/D converter voltage
input    : port - ADC port [0:1]
output   : DXn - store result in DXn [0 < n < SCR_DX_COUNT-1] (optional)
return   : ADC - voltage [0-5V]
```

GET BITS

```
use      : get bits [CXn]
brief    : show IO bit state summary
output   : CXn - store result in CXn [0 < n < SCR_CX_COUNT-1] (optional)
return   : IO bit states
```

GET NEXT

```
use      : get next
brief    : show time (s) until next scheduled activity
return   : time (s) or SYS_MAX_SLEEP_MS if no events pending
```

GET SER

```
use      : get ser <port>
brief    : read from serial port string until buffer full or end of input
input    : port - serial port [port 0:2]
return   : bytes read
```

GET TIME

```
use      : get time [u]
brief    : show system time
input    : u - output epoch time (optional)
return   : current date/time
```

GET VAR get var []

```
use      : get var [V] [key...]
brief    : display one or more variable values
input    : V - verbose output
input    : key - variable name (key)
return   : value of <key>, or list of key/value pair
          : values if if unspecified
```

Group [SET]

SET BAUD

```
use      : set baud <port> <bps>
brief    : set UART communication speed (bps)
input    : port - serial port [0:2]H|C|L]
input    : bps - rate (bps) [300,600,1200,2400,4800,9600,19200,38400]
return   : none
```

SET BITS

```
use      : set bits [id...]
brief    : set (HI) one or more digital IO bits
input    : id - bit number or name
```

```
#n : bit n (decimal)
    0:19 valves/acid pump/sea pump
    20:21 pump
    22:22 licor
    23:24 analog
    26:28 RS232 drivers
xn : bit mask n [0-FFFFFF] (hex)
an : ADC n 0:1
a* : all ADC
vn : valve n A&B
vnA|B : valve n A,B
v* : all valves
pn : DC pump n 0:3
pi,pr : air pump
pe,pw : seawater pump
pa : acid pump
ps : sample pump
pd : DC pumps (air, seawater)
px : solenoid pumps (acid, sample)
p* : all pumps
    l : licor DC power
    uh : host xcvr
    ul : licor xcvr
    uc : console xcvr
return : none
```

SET DEBUG

```
use : set debug [Q] [<op> <mask>]
brief : show/set/clear module debug enable flags
input : op - operator [-:clear +:set =:assign]
input : mask - hex bit mask
input : Q - quiet (suppress output)
return : debug flag settings
```

SET SER

```
use : set ser <port> <string>
brief : send serial string to specified serial port
input : port - serial port [0:2]H|C|L]
input : string - string to send
return : none
```

SET TIME get var []

```
use : set time YY[YY] MM DD hh mm ss
brief : set system time/date
input : YY[YY] - year [00-9999]
input : MM - month [01:12]
input : DD - day [01:31]
input : hh - hour [00:24]
input : mm - minute [00:59]
input : ss - second [00:59]
return : new time
```

SET VAR

```
use : set var <key> <val>
brief : set variable <key> to <val>
input : key - variable name
input : val - new value (must be appropriate type and format)
return : none
```

Group [CLR]

CLR BITS

```
use : clr bits [id...]
brief : clear (LO) one or more digital IO bits
input : id - bit number or name (see SET BITS)
return : none
```

Group [PULSE]

PULSE BITS

```
use      : pulse bits <dir> <delay_ms> bits [id...]
brief    : pulse one or more digital IO bits
input    : dir      - direction [L|0|H|1]
input    : delay_ms - pulse duration (ms)
input    : id       - bit number or name (see SET BITS)
return   : none
```

PULSE NBITS

```
use      : pulse nbits <dir> <ta> <tb> <n> bits [id...]
brief    : pulse one or more digital IO bits w/ specified duty cycle, number of cycles
input    : dir      - direction [L|0|H|1]
input    : ta       - pulse duration (dir, ms)
input    : tb       - pulse duration (ldir, ms)
input    : cycles   - pulse duration (ms)
input    : id       - bit number or name (see SET BITS)
```

Group [LI] (licor)

LI CAL

```
use      : li cal <id> [<spanc>]
brief    : perform calibration
input    : id       - cal ID [z:zero s:span s2:span2]
input    : spanc    - span concentration (required for span calcs)
return   : none
```

LI INIT

```
use      : li init
brief    : initialize licor
return   : none
```

LI NMEAS

```
use      : li nmeas <handler> <sample_n> <period> [options]
brief    : take multiple measurements, process using specified handler
          send output to specified destination(s).
input    : handler   - handler [a:average b:baseline r:]
input    : sample_n  - number of records to process
input    : period    - licor output period (s) [>=0.5]
input    : options   - options
                    O<destination> : H:host C:console L:log A:all
                    R<rec_type>    : character
                    C<rec_context> : character
                    T<tmout>       : timeout (ms) default 1.1*sample_n*period
return   : record (varies by handler)
```

LI OR

```
use      : li or <period>
brief    : set licor output rate
input    : period    - output period (s) [period >=0.5]
return   : none
```

LI TM

```
use      : li tm [options]
brief    : take one or more measurements using specified delay.
          Send output to specified destination(s)
input    : options
          O<destination> : H:host C:console L:log A:all (default:C)
          R<rec_type>    : character (default:U)
          C<rec_context> : character (default:?)
          N<samples>     : number of samples (default:1)
          D<delay>       : delay between samples (ms) (default:0)
return   : record
```

Group [SCR] (script)

SCR DEBUG

```
use      : scr debug +|- <id>
brief    : enable/disable debug output for specified script
input    : id - script name or handle
return   : none
```

SCR DELAY

```
use      : scr delay <time_ms> <id>
brief    : delay specified period before executing next command
input    : time_ms - delay time (ms)
input    : id      - script name or handle
return   : none
```

SCR END

```
use      : end
brief    : end of script [only valid in script context]
return   : none
```

SCR JOG

```
use      : scr jog <id>
brief    : single-step the specified script [enters manual mode]
input    : id - script name or handle
return   : none
```

SCR LOAD

```
use      : scr load [o<+/->] <file> [<file>...[o<+/->]...]
brief    : load specified script(s) into next available handle.
input    : file - script file name
input    : o      - enable/disable overwrite for files that follow (may use more than once)
return   : confirmation or error message
```

SCR RUN

```
use      : scr run <id>
brief    : run the specified script.
input    : id - script name or handle
return   : none
```

SCR SCH

```
use      : scr sch <period_ms> [<sync>] <id>
brief    : schedule script, optionally synchronize (run)
input    : period_ms - schedule period (ms)
input    : sync      - sync now [Y|N]
input    : id        - script name or handle
return   : none
```

SCR SHOW

```
use      : scr show [<id>...]
brief    : print script parameter summary for one or more scripts
input    : id - script name or handle or '*'
return   : none
```

SCR SYNC

```
use      : scr sync <time_ms>
brief    : start script after optional delay
           [shifts scheduled start time]
input    : time_ms - delay time (ms) (optional)
return   : none
```

SCR UNLOAD

```
use      : scr unload <id>
brief    : unload specified script.
input    : id - script name or handle
return   : none
```

Group [MAC] (macro)

Use notes

MDIO and MMEAS separate IO manipulation from measurement processing. MDIO may be used to set up valves and pumps with specified delays. MMEAS is used to process a set of licor measurements, using specified parameters. They provide fine-grained control of cycle operations, as well as simple presets (macros) that encapsulate several IO operations and timing, with minimal parameter overrides.

The use of M macros are deprecated, but they currently available for testing. MDIO, MMEAS should be used for production scripts

M equivalents are implemented using mdio and mmeas:

PURGE: mdio purge FILL: mdio fill ACID: mdio acid ZERO: mdio zero MEAS: mdio sample mmeas prebl sample postbl mdio measx AIR: mdio air mmeas air mdio measx STD: mid std mmeas std mdio measx SZERO: mdio szero REST: mdio rest

MDIO

```
use      : mdio [ <preset> d<delay> vd<dir> vt<ton> ph<ton> pl<toff>
              pn<cycles> <vset> <pump> +sea -sea +air -air +li -li ]
brief    : assert digital IO w/ optional pre/post delays
          : before/during/after measurement
input    : d      - delay time (msec)
input    : vd     - valve dir:H|L
input    : vt     - valve pulse time (msec)
input    : ph,l   - pump pulse time (msec)
input    : pn     - pump cycles
input    : mt     - valve pulse time (msec) [1]
input    : mh,l   - pump pulse time (msec) [1]
input    : mn     - pump cycles [1]
input    : preset - macro presets: fill air purge zero szero std  stdfill sample measx rest
input    : pump   - cycle pump: psample pacid (must set pn before pump)
input    : vset   - switch valve set: iofill ioair iopurge iozero ioszero iostd iostdfill iomeas iorest
input    : +/-sea - seawater pump on/off
input    : +/-air - air pump on/off
input    : +/-li  - licor on/off
```

```
[1] Note: mt,mh,m1,mn work the same as vt,ph,pl,pn except that
        - they only apply to presets (macros)
        - per-macro defaults used if they are unset (see cli_impl.h)
        Like vt, ph, etc, they are also sticky (per command line); set to zero to restore defaults
```

MMEAS

```
use      : mmeas [r<rtype> c<ctx> p<period> h<handler> o<output>
                  t<mtime> n<mcount> w<cal_wr> d<delay> meas <preset> ]
brief    : licor measurement cycle
input    : rtype  - record type (char) [UVBRS?]
input    : ctx    - measurement context (char) [UZSMasz?]]
input    : mtime  - measurement time (msec)
input    : mcount - measurement count (alternative to mtime)
input    : handler - measurement handler (A:average|B:baseline)
input    : period - licor sample period (sec >=0.5)
input    : output - output (A:all H:host C:console L:log)
input    : cal_wr - enable/disable writing calibration to licor [+|-]
input    : delay  - delay (msec)
input    : meas   - take measurement using current settings
input    : preset - packaged settings: zero air std sample prebl postbbl zcal scal
```

M macros [deprecated]

MACID

```
use      : macid [n<cycles> h<hi_msec> l<lo_msec>]
brief    : inject acid
input    : cycles - number of pump cycles
input    : cycle_msec - cycle time (msec)
```

MAIR

```
use      : mair [v<valve_pulse_msec> p<pump_on_msec> m<meas_msec>]
brief    : take air measurement
input    : valve_pulse_msec - valve pulse time (msec)
input    : pump_on_msec    - air pump on time (msec)
input    : meas_msec      - measurement time (msec)
```

MFILL

```
use      : mfill [v<valve_pulse_msec> s<sea_pump_on_msec> n<sample_pump_cycles>
                  h<sample_pump_hi_msec> l<sample_pump_lo_msec>]
brief    : fill sample chamber
input    : valve_pulse_msec - valve pulse time (msec)
input    : pump_on_msec    - sea pump on time (msec)
input    : sample_pump_cycles - sample pump cycles
input    : sample_pump_cycle_msec - sample pump cycle time (msec)
```

MMEAS_DEP

```
use      : mmeas_dep [v<valve_pulse_msec> r<rate> a<bline_msec> b<bline_msec> m<meas_msec>]
brief    : sparge, take measurement
input    : valve_pulse_msec - valve pulse time (msec)
input    : rate             - licor output period (sec >=0.5)
input    : meas_msec        - measurement time (msec)
input    : bline_msec       - baseline measurement time (msec)
```

MPURGE

```
use      : mpurge [v<valve_pulse_msec> p<pump_on_msec>]
brief    : purge sample
input    : valve_pulse_msec - valve pulse time (msec)
input    : pump_on_msec    - air pump on time (msec)
```

MREST

```
use      : mrest [v<valve_pulse_msec>]
brief    : put valves in rest position
input    : valve_pulse_msec - valve pulse time (msec)
```

MSTD

```
use      : mstd [v<valve_pulse_msec> p<pump_on_msec> m<meas_msec>]
brief    : measure std gas
input    : valve_pulse_msec - valve pulse time (msec)
input    : pump_on_msec    - air pump on time (msec)
input    : meas_msec       - measurement time (msec)
```

MSZERO

```
use      : mstd [v<valve_pulse_msec> p<pump_on_msec>]
brief    : short zero condition
input    : valve_pulse_msec - valve pulse time (msec)
input    : pump_on_msec    - air pump on time (msec)
```

MZERO

```
use      : mstd [v<valve_pulse_msec> p<pump_on_msec> e<vent_pulse_msec>]
brief    : zero condition
input    : valve_pulse_msec - valve pulse time (msec)
input    : pump_on_msec    - air pump on time (msec)
input    : vent_pulse_msec - vent pulse time (msec)
```