MBARI Carbon WG Console Help

Command syntax help is organized using the following topics:

Group	Description	Links
REG	built-in register variables	Group REG
QUEUE	sample queue	Group QUEUE
FSYS	file system	Group FSYS
LOG	logging	Group LOG
DIAG	diagnostics/debug	Group DIAG
GET	inspect system state/status	Group GET
SET/CLR	change configuration/state/IO	Group SET
PULSE	momentary IO cycling	Group PULSE
TEST	test IO state (store result in register)	Group TEST
SCR	script control	Group SCR
APP_LI	application-specific: licor control	Group APP_LI
APP_EX	application-specific: experiment control	Group APP_EX
REF	reference	Group REF

General syntax guidelines

- indicates an option named foo, which has yet to be defined
- In the context of a command line use note, arguments in square brackets are optional [ ] but may contain required elements [op *] means you may use 'op ' but not just 'op' or ''*
- In the context of command notes, options in square brackets may indicate set of literal values [CXn|*] or [=,–,++,+=.-=.*=,/=,%=]
- Options with ellipses '…' indicates that that the preceding option may be repeated […] is one or more options
- Options don't use dashes (H not -H)
- Except where it is ambiguous, there is no space between option arguments and values e.g. X1.23, FOO3
- A colon is used to disambiguate options and arguments; e.g. a command has options S and S2 with numeric values - S:123 S2:1.34
- * is a wildcard, signifying all/any

Group REG

(registers)

The system implements a set of multi-purpose variables (registers) that may be used in scripting and expressions.
Register variables include

```
CREG : 8 x 32-bit signed integer counters
DREG : 8 x double
FREG : 32 x flags (boolean single bit)
```

Some commands allow values to be stored in register variables, and register variables may be used in script boolean expressions.
Register variables are global in scope; they are visible to all scripts and users.
This makes it possible for scripts to communicate using variables.
Users must protect variables from accident modification through concurrent use.

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

(sample queue management)

The system implements two queues, a record queue and a measurement queue. These may be used to aggregate related measurements and records, for example the most recent measurements or record.
The queue API allows users and client software to inspect and/or delete contents.
This can be useful for an operation model in which a client periodically requests the latest measurements or records.

QCOUNT

```
use    : qcount i<id>
brief  : return queue entries
input  : id    - queue ID 0:meas 1:cycle
return : number of queue entries
```

QCLEAR

```
use    : qclear i<id>
brief  : delete all queue entries
input  : id    - queue ID 0:meas 1:cycle
return : none
```

QPEEK

```
use    : qpeek i<id> [<first>] [<n>]
```

```
brief  : examine queue entries (don't delete)
input  : id    - queue ID 0:meas 1:cycle
input  : first - starting queue entry
input  : n     - number of entries or '\*'
input  : dest  - output destination
                 C:console
                 H:host
                 L:log
input  : fmt   - output formatting/content
                 P  : pretty
                 XX : pretty hex
                 X  : hex
                 X* : pretty hex + hex
                 B  : base85
                 C  : CSV
                 R  : raw
                 H  : header
                 D  : data
                 V  : verbose
return : decoded queue entries
```

QPOP

```
use    : qpop i<id> [Q] [OUT:<dest>[,<fmt>] <n>
brief  : delete and optionally show next (oldest) n entries from qu
input  : id   - queue ID 0:meas 1:cycle
input  : n    - number of entries or '*' for all
input  : Q    - suppress output (just delete)
input  : dest - see QPEEK
input  : fmt  - see QPEEK

return : oldest n entries in the queue (or none if Q specified)
```

QSHOW

```
use    : qshow [id]
brief  : print queue summary
input  : id    - queue ID 0:meas 1:cycle
return : queue summary
```

Group FSYS

(file system)

File system commands provide simple capabilities for manipulating files and viewing contents.

CAT

```
use    : cat [T<n>|H<n>] <file>
brief  : show file contents
input  : file - file to show
input  : H<n> - show first n bytes
input  : T<n> - show last  n bytes
return : file contents
```

CFGEX

```
use    : cfgex <file>
brief  : export current configuration to file
input  : file - destination file [1]
return : none

[1] : file will be overwritten if it exists
```

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
```

DSTAT

```
use      : DSTAT [<dir>...]
```

```
brief    : show file system information
input    : vol - volume
return   : file system summary
examples :

# Show summary for SD card top directory
dstat
```

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]
```

LS|DIR

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

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
```

UPLOAD

```
use    : upload [no-eol] <dest>
brief  : upload ASCII file to root directory
input  : dest   - file name [1]
         no-eol - disable auto EOL translation [2]
Notes
[1] 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

[2] automatically converts line ends by default:
     replace \r\r, \n\n with \r\n\r\n
     replace .\r., .\n. with .\r\n.
     add \r\n to end of file if it doesn't exist

[3] some terminal emulators may not support direct ASCII upload

return : none
```

XCAT

```
use: xcat [h v<n>] [r:<rspec>] <file>
brief: Show text file regions, specified by time, lines, offset
input: h       -  help message
input: v[<n>]  -  verbose output
input: <file>  -  file name
input: <rspec> -  range specifier
                  comma separated list of range modes and values:
```

```
                    <start_mode>,<val>,<end_mode>,<val> [1,2]
              <mode> : range type [blims][+-]
                       b - byte mode
                       l - line mode
                       i - time mode (ISO8601)             [3]
                       m - time mode (epoch msec)          [3]
                       s - time mode (epoch sec)           [3]
                       - - relative to end of file         [4]
              <val> : range value
                      line/byte count (index >=0)
                      timestamp (epoch sec/msec or ISO8601)
Notes
[1] range specifiers optional; use empty field (no space) if unspec
[2] valid end modes include [blims]; others ignored. Start/end time
[3] file lines must start with specified timestamp (leading space O
[4] line/byte start modes only
     -  Defaults: line mode, start at beginning of file, show all
     -  Use CR or CTRL-D to interrupt

examples:
 # show last 10 lines of sys2000.000
    xcat r:-,10 sys2000.000
 # show first 10 lines of sys2000.000
    xcat r:,,,10 sys2000.000
 # show first 10 lines of sys2000.000 >= 2020-03-28T00:00:00
    xcat r:i,2020-03-28T00:00:00,l,10 sys2000.000
 # show first 10 lines of foo.csv >= 1592974762 (Tue Jun 23 21:59:2
    xcat r:s,1592974762,l,10 foo.csv
 # show 100 bytes starting at line 10 sys2000.000
    xcat r:l,10,b,100 sys2000.000
```

Group LOG

(logging)

Access log entries and metadata.

LOG

```
use    : log <msg>
brief  : add annotation to syslog
```

```
input  : msg - message to log
return : none
```

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, (
return : none
```

LSHOW

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

LCAT

```
use    : lcat <file> [s<start>] [c<count>] [l<count>] [out:<fmt>]
brief  : show selected log records using specified formatting
input  : file - log file name (e.g. dat2003.000)
input  : s   - start record
               start : start record index (zero-based)
input  : c   - number of records
                count : number of records
input  : l   - last <count> records
input  : fmt - output formatting
               P  : pretty
               XX : pretty hex
                X : hex
               X* : pretty hex + hex
               B  : base85
               C  : CSV
               R  : raw
               H  : header
               D  : data
               V  : verbose (show record index/offset)
default : fmt:PDHV start=0 c=-1 (all records)
```

```
return  : none
```

LLESS

```
use    : lless <file> [<out:[pbcrx[x*]hdv]>]
brief  : interactive log inspection
input  : file - log file name (e.g. dat2003.000)
input  : out  - output format control (may be changed interactively
return : none
summary: There are three types of inputs:
         Movement : navigating records
         Command  : search, output settings
         Session  : control session, help, etc.

         Movement :
           First/Last [ g <    ] / [ G >  ]
           Next/Prev  [ dD<CR> ] / [ uU   ]
           This       [  <SPC> ]
           Jump +/-   [ j<n>   ] / [ k<n> ]

         Session  :
           Help [h H ?]   Quit [q Q x X]

         Command  :
           Enter cmd mode [ / ]
           Search -
                   tm<msec> ti<iso1806>
                   r<rtid> rm<mtid> mm=<mmid>
                   i<record index>
                   Next match [n]
                   Prev match [N]
           Output -
                   out:<fmt> where fmt is one or more of
                   P  : pretty
                   XX : pretty hex
                   X  : hex
                   X* : pretty hex + hex
                   B  : base85
                   C  : CSV
                   R  : raw
                   H  : header
                   D  : data
                   V  : verbose (show record index/offset)
```

LINFO

```
use    : linfo h out:prcieh <file...>
brief  : show log record summary
input  : file - log file name (e.g. dat2003.000)
return : summary
note   : interrupt with CR or CTRL-D
summary: options:
           h - help
         out - specify output options
               p - pretty format (default)
               r - record counter (progress)
               c - CSV format
               e - epoch time (posix, CSV only)
               i - ISO8601 time (CSV only)
               h - include CSV header (CSV only)
```

LSEQ

```
use    : lseq [D]
brief  : show next sequence number  (from file)
input  : D - data log
return : sequence number
```

Group DIAG

(diagnostics/debug)

Diagnostics and system utilities

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 spe
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
examples :
  >>debug =0
  SYS_MDFLAGS=x00000000
  >>debug +4
  SYS_MDFLAGS=x00000004
  >>debug +8
  SYS_MDFLAGS=x0000000c
  >>debug -8
  SYS_MDFLAGS=x00000004
```

PMEM

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

PRINT, PRINTLN

```
use      : print <fmt>  [<var>...]
brief    : print formatted messages to console using variable expans
           character support.

           fmt is a quoted format string, which may include , escape
           (with $ prefix), printf formats

           var are variables (without $ prefix)
           println variant automatically appends CRLF to string

return   : formatted console message (upper case)
examples :

 # set CX reg
  >>CREG CX0 = 20

 # print CX using default (hex) format
  >>PRINT "$CX0\r\n"
  X00000014

 # print CX using alternative (decimal) format
  >>PRINT "%ld\r\n" CX0
  20
```

ECHO

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

```
note   : using WITH_UI_ECHO
```

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

```
[deprecated; use PRINT, PRINTLN is recommended]

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

Group GET

(inspection)

Utilities for inspecting system status and state

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] (optio
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] (optiona
return : IO bit states

[1] - ADC2 does not have an enable bit, always off
```

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 PRINT

```
use    : get print [DV]
brief  : show value of debug and verbose print variables
         used to configure DPRINT and V*PRINT macros
return : value of debug and/or verbose print variables
```

GET SER

```
use    : get ser <port>
brief  : read from serial port string until buffer full or end of i
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

```
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/CLR

(change system hardware state)

Utilities for changing system hardware state parameters.
The set API provide full, fine-grained control of system IO.
The mdio API is application-specific, used for pCO2/DIC sampling.

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/sample pump
               20:21 pump
               22:22 licor
               23:24 analog
               26:28 RS232 drivers
       x<n>  : bit mask n [0-FFFFF] (hex)
```

```
        a<n>  : ADC n 0:1
        a*    : all ADC
        vn    : valve n A&B
        vnA|B : valve n A,B
        v*    : all valves
        p<n>  : DC pump n 0:5
        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 UART xcvr
        ul    : licor UART xcvr
        uc    : console UART xcvr
return : none
```

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
```

SET PRINT

```
use    : get print [DV]
brief  : set value of debug and verbose print variables
         used to configure DPRINT and V*PRINT macros
         DPRINT is enabled if !=0
         VPRINT is enabled if !=0, and allows levels (>0)
return : none
```

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

```
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 PULSE

(GPIO momentary switching)

Momentarily assert one or more digital IO bits (in either direction)

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
input  : dir    - direction [L|0|H|1]
input  : ta     - pulse duration (dir, ms)
input  : tb     - pulse duration (!dir, ms)
input  : cycles - pulse duration (ms)
input  : id     - bit number or name (see SET BITS)
```

Group TEST

(logical IO bit test)

Logically test one or more IO bits, and store the result in a register. This may be used to implement flow control in a script context.

TEST BITS

```
use    : test bits [Q] [ MCXn BCXn OCXn ]  [M<mask>...] [<id>...]
brief  : test one or more IO bits.
input  : Q      - quiet mode (no console output, for scripting)
input  : BCXn   - store bits in CXn
input  : MCXn   - store mask in CXn
input  : OCXn   - store output (mask&bits) in CXn
input  : Mmask  - add <mask> (hex) to bitmask [1]
input  : id     - bit number or name (see SET BITS) [1]
return : none

[1] - may be included more than once (all values OR'd)\

Examples :
# Using test bits for script flow control
# clear all IO bits (except for tranceivers)
```

```
  clr bits p* v* a* l
# set ADC bit
  set bits a*
# store current IO state in CX
# store ADC bit state in CX1
  test bits a* MCX0 OCX1
# compare bit state vs mask
  if [ CX0 == CX1 ] goto pass
# test failed
   println "set bits a [ERR / $CX0 / $CX1]"
   goto cont
   :pas
# test passed
  println "set bits a* [OK]"
  :cont
```

Group SCR

(scripting)

Commands used for managing scripts

SCR SDEBUG

```
use    : scr sdebug +|- <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
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 RESET|KILL

```
use    : scr reset <id>
brief  : reset the specified script. Does not change hardware states
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 APP_LI

(licor)

Application specific : Licor gas analyzer control API

LI CAL

```
use      : li cal <id> [ D<delay>]
brief    : perform calibration
input    : id   - cal ID, where <id> may be
               z        [zero]
               s:d.ddd  [span]
               s2:d.ddd [span2]
input    : delay - extend delay (in addition to 10s max implemented
examples :
  li cal s:452.1  [S cal, span concentration 452.1]
  li cal s2:23.05 [S2 cal, span concentration 23.05]
  li cal s:0      [S cal, use LI_SPANC configuration for span conce

return   : none
```

LI INIT

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

LI OR

```
use    : li or <period>
brief  : set licor output rate
input  : period - output period (s) [period >=0.5]
return : non
```

LI STAT

```
use      : li stat [i<indent> v] <tag>...]
brief    : output licor status to console (formatted xml)
input    : indent - number of space to indent
input    : v      - verbose output
input    : tag    - output contents of first occurrence of tag
return   : none
examples :
 // show <rs232>...</rs232> and <cfg>...</cfg>
    li stat rs232 cfg
 // show all, indent 8 spaces
    li stat i8
```

Group APP_EX

(pCO2/DIC experiment)

Application specific : Measurement control macros tailored to control pCO2 and DIC measurements using Licor gas analyzer.

About MDIO and MMEAS APIs

MDIO and MMEAS separate IO manipulation from measurement processing.

MDIO may be used to set up valves and pumps with precision timing (~10 ms resolution). MMEAS is used to process a set of licor measurements, using specified parameters and timing.

They provide fine-grained and flexible control of measurement operations, including macros that encapsulate multiple operations and their parameters, with minimal parameter overrides.

MDIO includes a compact syntax for composing IO sequences.

MMEAS uses a measurement profile buffer that persists between calls (the

operation profile); this buffer is always used when executing a measurement.

There are a set of user-defined profile buffers, and read-only preset profiles. The USR and LOAD (or TS) MMEAS options are used to inialize the operation profile, apply optional overrides, and/or inspect the configuration before performing the measurement.

The TS (take sample) option automically runs specified profiles after any options are processed.
The LOAD option requires the RUN option, e.g.

```
ts:air,b2000
```

is equivalent to

```
load:air,b2000,run
```

MDIO

```
use    : MDIO [ options... ]

brief  : Assert digital IO and macros w/ optional delay between ope
         Multiple options may be provided.

Options:

 <macro>   - Run macro (commonly used IO sequences, see notes)

 <preset>  - Actuate pre-defined valve sets or pumps (see notes)

 PHL, PLH  - pulse IO bit(s) HI/LO, LO/HI (multiple cycles, e.g. pu
             use: phl:<iobits,ton,toff,cycles,[post-delay]>

 SHL, SLH  - pulse IO bit(s) HI/LO, LO/HI (single cycle, toff=0 e.g
             use: shl:<iobits,ton,[post-delay]>

 IOH, IOL  - set IO bit(s) HI/LO (i.e. ON/OFF)
             use: ioh:<iobits,[post-delay]>

 D         - delay for specified time (msec)
             use: d<n> (e.g. d5000) or d$cx<n> (e.g. d$cx3)
```

```
  PT        - preset valve pulse time (msec)
              Applies only to IO presets
              use: pt<n> (e.g. vt200)

  PH        - preset pump pulse ton (msec)
              Applies only to IO presets
              use: ph<n> (e.g. ph200)

  PL        - preset pump pulse toff (msec)
              Applies only to IO presets
              use: pl<n> (e.g. pl300)

  PN        - preset pump pulse cycles
              Applies only to IO presets
              use: pn<n> (e.g. pn15)

  MT        - macro valve pulse time (msec)
              Applies only to macros
              use: mt<n> (e.g. mt100)

  MH        - macro pump pulse ton (msec)
              Applies only to macros
              use: mh<n> (e.g. mh150)

  ML        - macro pump pulse toff (msec)
              Applies only to macros
              use: ml<n> (e.g. ml300)

  MN        - macro pump pulse cycles
              Applies only to macros
              use: mn<n> (e.g. mn20)

  +V,-V     - enable/disable verbose output

  SHOW      - show state and preset/macro names

 where:

   iobits     - IO bits (use SET BITS syntax, separated by '/')
   ton        - pulse ON time  (msec)
   toff       - pulse OFF time (msec)
   cycles     - pulse cycles
   post-delay - post-opertion delay (msec)
   <n>        - integer number

## <preset> Values (IO Presets) [2]:

   +/-AIR       - air pump on/off
   +/-SEA       - seawater pump on/off
   +/-LI        - licor power on/off
   APUMP        - run acid pump
   SPUMP        - run spump
   +/-IOPURGE   - purge vset
```

```
   +/-IOFILL    - fill vset
   +/-IOZERO    - zero vset
   +/-IOZVENT   - zero vent vset
   +/-IOZCAL    - zcal vent vset
   +/-IOSZERO   - szero vset
   +/-IOSZVENT  - szero vent vset
   +/-IOSAMPLE  - sample measurement vset
   +/-IOAIR     - air measurement vset
   +/-IOSTD     - standard measurement vset
   +/-IOSTDFILL - standard fill vset
   +/-IOREST    - rest vset
   +IOEXCERn    - exercise step n[1:8] vset (+ only)
   +/-IOSCAL    - scal vset
   +/-IOSCALGAS - scal gas vset

## <macro> Values

   FILL   - Fill volume
   PURGE  - Empty volume
   ACID   - Inject acid
   ZERO   - Zero path
   ZCAL   - Zero calibration
   SZERO  - SZero calibration
   SAMPLE - Sample measurement
   AIR    - Air measurement
   SCAL   - Standard calibration
   STD    - Standard measurement
   MTERM  - Turn of pump and licor
   REST   - Put valves in rest state
   VEX    - Exercise valves

 Examples:

    IO preset:

          # Run purge macro (set up valve IO, w/ pump, delays, etc.)
           mdio purge

          # Set valve IO preset only
           mdio +iopurge

          # Clear valve IO preset only
           mdio -iopurge

         Using mdio with mmeas:

          # Use setup macro prepare for sample measurements
             mdio sample

          # Take measurements
             mmeas mmpbl prebl mms sample mmpob postb

          # Use mterm macro to turn off air pump and licor
          # after measurement
```

```
              mdio mterm

    User defined IO sequence:

          # Execute the following sequence
          # - pulse v1a, v3a,b ON
          # - wait 5s
          # - pulse acid pump 15 cycles (300 ms ON, 200 ms OFF)
          # - turn on air pump
          # - wait 10s
          # - turn air pump OFF
          # - set pump cycles=10 and run sample pump with default pu
          # - put IO in rest configuration

            mdio vhl,v1a/v3,5000 phl,pa,300,200,15 +air d10000 -air

Notes:

     Operations are evaluated and executed left to right.

     Command and options are case-insensitive

     Command length limited by UI_CMD_BUF_BYTES (console context)
     or SCR_LINE_BUFFER_BYTES (in script context). Use multiple line
     for long sequences.

 [1] options pt, ph, pl, pn, mt, mh, ml, mn
     - are sticky (per command line), may set multiple times
     - use a default value if unset
     - defaults used if unset (see cli_impl.h)

 [2] pump cycles (pn) must be set >0 to enable pump IO presets e.g.
     - pn, mn are sticky (per command line)
     - may be set multiple times on same line
     - use phl/plh to actuate multiple pumps simultaneously
```

MMEAS

```
 use    : mmeas <cmd>:[options]

 brief  : Configure and/or execute one or more licor measurement pro

 Commands (<cmd> values):

  CYC     -  Control measurement cycle parameters
             use: CYC:[option,...]
             Options:
              <RTID>   - record type ID
              <RMID>   - record measurement ID
```

```
             RST      - reset measurement buffer (resets rtid, rmid

             Example:
             Reset measurement buffer, Set record type to RTZ0, rec
             to RMSAMPLE
              cyc:rst,rtz0,rmsample


USR<n>  -  Configure user-defined measurement profile (sticky)
             use: USR<n>:[option,...]
             Options:
               <MEAS_PRESET>    - copy measurement preset (optional
               <MEAS_PARAMETER> - measurement parameters
               <RTID>
               <RMID>
               <MMID>           - record MMID (logged value)
               <OPID>           - operation MMID
                                  If MMSCAL, MMZCAL, enables/disabl
                                  Licor if cal_wr=='+'
             Example:
             Set user measurement profile, based on AIR preset;
             overrides blanking time and number of samples.
              usr2:AIR,b2000,n13

             Change USR2 licor period to 1.0
              usr2:p1.0

LOAD,TM -    Load measurement profile(s), optionally apply override
             LOAD will load and configure a profile, but requires e
             This enables load/inspect without running.
             TM automatically runs specified profile(s) immediately

            use: LOAD:[option,...] or TM:[options]

            Options:
             <MEAS_PRESET>    - load measurement preset name
             USR<n>           - load user-defined profile
             <MEAS_PARAMETER> - measurement parameters (overrides)
             RUN              - run current profile

            Example:
            Load and run SAMPLE preset, override number of licor sa
             mmeas meas:sample,n14,run
             mmeas tm:sample,b2000,air

OUTR    -    Output record buffer
               use: outr:[CHLQ]*,[PBRCXXHDV]*

OUTM    -    Output last measurement
               use: outm:[CHLQ]*,[PBRCXXHDV]*

OUT<n> -     Output record measurement[0<=n<12]
               use: out[0-9]+[0-1]*:[CHLQ]*,[PBRCXXHDV]*
```

```
               where

                [CHLQ] - destination flags
                 C : console
                 H : host
                 L : log
                 Q : queue

                [PBRCXX*HDV] - format flags (optional, for console o

                 P  : Pretty
                 B  : Base85 (ASCII encoding, more efficient than uu
                 C  : CSV
                 R  : raw
                 X  : hex
                 XX : PrettyHex
                 X* : hex+PrettyHex
                 H  : header
                 D  : data
                 V  : enable verbose content (if any)

               Examples:
                  mmeas outm        output last meas to console (Pre
                  mmeas outm:c,b    output last meas to console (Bas
                  mmeas outm:c,pxx  output last meas to console (Pre
                  mmeas outr:q      output record to (cycle) queue
                  mmeas outm:q      output measurement to (meas) que
                  mmeas out4:q      output 5th (i.e. index=4) measur

 SHOW   -  Inspect record buffer summary and measurement profiles
             use: show:[A*BHOPRU]

             where

              [A*BHOPRU] - destination flags

               A,* : all
               B   : measurement buffer
               H   : help
               O   : current operation profile
               P   : preset profiles
               R   : measurement record
               U   : user-defined profiles


Options:

 ## <RTID> Values
 RTZ0          - set record type to default
 RTX<x>        - set record type (user defined)
                 use: rtx<x> (e.g. rtx3e5)
 ## <RMID> Values
 RMSAMPLE,RMS  - set record measurement type sample
 RMSCAL,RMSC   - set record measurement type scal
```

```
 RMZCAL,RMSZ   - set record measurement type zcal
 RMX<x>        - set record measurement type (user defined)

 ## <MMID> Values
 MMAIR,MMA     - set measurement ID air
 MMSTD,MMST    - set measurement ID standard
 MMZERO,MMZ    - set measurement ID zero
 MMSAMPLE,     - set measurement ID sample
 MMPRES,MMPRS  - set measurement ID pre-standard
 MMPOSTS,MMPOS - set measurement ID post-standard
 MMPREZ,MMPRZ  - set measurement ID pre-zero
 MMPOSTZ,MMPOZ - set measurement ID post-zero
 MMPREB,MMPRB  - set measurement ID pre-baseline
 MMBLINE,MMB   - set measurement ID baseline
 MMPOSTB,MMPOB - set measurement ID post-baseline
 MMZCAL,MMZC   - set measurement ID z-calibration
 MMSCAL,MMSC   - set measurement ID s-calibration
 MMX<x>        - set measurement ID user-defined

## <MEAS_PARAMETER>  Values

 P<n>          - set licor sample period (decimal sec, 0.0 or >=0.5
                 use: p<f> (e.g. p0.5)

 T<n>          - set measurement time
                 use: t<n> {e.g. t30000}

 N<n>          - set measurement sample count
                 use: n<n> {e.g. n16}

 B<n>          - set blanking period (msec)
                 use: b<n> {e.g. b2000}

 W<n>          - enable/disable calibration value write to instrume
                 use: w[+-] (e.g. w+)

 D<n>          - delay for specified time (immediate, msec)
                 use: d<n> {e.g. d12500}

## <MEAS_PRESET> Values

 <preset>   - use pre-defined measurement parameters
              FILL   - Fill the chamber
              PURGE  - Purge gas
              ACID   - Inject acid
              ZERO   - Zero gas paths
              ZCAL   - Zero calibration
              SCAL   - Standard calibration
              SZERO  - Zero standard paths
              SAMPLE - Measure sample
              AIR    - Measure air
              STD    - Measure standard

Examples:
```

```
  # reset measurement buffer (e.g. at start of cycle)
  # and set record measurement type to RMSAMPLE
   mmeas cyc:rst,rms

  # take sample measurement, output measurement to console
   mmeas ts:sample outm:c,pdh

  # take air sample (using mmid=99) and output record to cycle queue
   mmeas ts:air,mm99 outr:q

  # load user-defined profile and show measurement info
   mmeas load:usr3 stat:OB

  # output record to console, in pretty and pretty hex format, with
   mmeas outr:c,pxxdh

  # output 5th (i.e. index 0:4) measurement to (meas) queue
   mmeas out4:q

  # output last measurement to console (Pretty)
  mmeas outm

  # output record to console (Base85)
  mmeas outm:c,b

Notes:

     Operations are evaluated and executed left to right.

     Command and options are case-insensitive

     Command length limited by UI_CMD_BUF_BYTES (console context)
     or SCR_LINE_BUFFER_BYTES (in script context). Use multiple lines
     for long sequences.
```

Group REF

(reference)

Reference information

IO bit indices

Name	bit	hex	binary
SV0A	00	0x00000001	0000 0000 0000 0000 0000 0000 0000 0001
SV0B	01	0x00000002	0000 0000 0000 0000 0000 0000 0000 0010
SV1A	02	0x00000004	0000 0000 0000 0000 0000 0000 0000 0100
SV1B	03	0x00000008	0000 0000 0000 0000 0000 0000 0000 1000
SV2A	04	0x00000010	0000 0000 0000 0000 0000 0000 0001 0000
SV2B	05	0x00000020	0000 0000 0000 0000 0000 0000 0010 0000
SV3A	06	0x00000040	0000 0000 0000 0000 0000 0000 0100 0000
SV3B	07	0x00000080	0000 0000 0000 0000 0000 0000 1000 0000
SV4A	08	0x00000100	0000 0000 0000 0000 0000 0001 0000 0000
SV4B	09	0x00000200	0000 0000 0000 0000 0000 0010 0000 0000
SV5A	10	0x00000400	0000 0000 0000 0000 0000 0100 0000 0000
SV5B	11	0x00000800	0000 0000 0000 0000 0000 1000 0000 0000
SV6A	12	0x00001000	0000 0000 0000 0000 0001 0000 0000 0000
SV6B	13	0x00002000	0000 0000 0000 0000 0010 0000 0000 0000
SV7A	14	0x00004000	0000 0000 0000 0000 0100 0000 0000 0000
SV7B	15	0x00008000	0000 0000 0000 0000 1000 0000 0000 0000
SV8A	16	0x00010000	0000 0000 0000 0001 0000 0000 0000 0000
SV8B	17	0x00020000	0000 0000 0000 0010 0000 0000 0000 0000
SV9A	18	0x00040000	0000 0000 0000 0100 0000 0000 0000 0000
SV9B	19	0x00080000	0000 0000 0000 1000 0000 0000 0000 0000
PUMP0_DC	20	0x00100000	0000 0000 0001 0000 0000 0000 0000 0000
PUMP1_DC	21	0x00200000	0000 0000 0010 0000 0000 0000 0000 0000
PUMP2_DC	22	0x00400000	0000 0000 0100 0000 0000 0000 0000 0000
PUMP3_DC	23	0x00800000	0000 0000 1000 0000 0000 0000 0000 0000
LICOR_DC	24	0x01000000	0000 0001 0000 0000 0000 0000 0000 0000
ADC0	25	0x02000000	0000 0010 0000 0000 0000 0000 0000 0000
ADC1	26	0x04000000	0000 0100 0000 0000 0000 0000 0000 0000
ADC2	27	0x08000000	0000 1000 0000 0000 0000 0000 0000 0000
XCVR_HOST	28	0x10000000	0001 0000 0000 0000 0000 0000 0000 0000
XCVR_LICOR	29	0x20000000	0010 0000 0000 0000 0000 0000 0000 0000
XCVR_CONS	30	0x40000000	0100 0000 0000 0000 0000 0000 0000 0000