

# 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
- \* TEST test IO
- \* PULSE momentary IO cycling
- \* LI licor control
- \* SCR script control
- \* MAC macros
- \* REF reference

## General syntax guidelines

---

- <foo> 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 [ <foo> <bar> ] but may contain required elements [op <var>] means you may use 'op <var>' but not just 'op' or '<var>'
- In the context of command notes, options in square brackets may indicate set of literal values [CXn|\*] or [=,--,
- Options with ellipses '...' indicates that the preceding option may be repeated [<foo>...] is one or more <fo
- 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, F003
- A colon is used to disambiguate options and arguments; e.g. a command has options S and S2 with numeric values - S
- \* is a wildcard, signifying all/any

---

## 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 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 '*'
return   : decoded queue entries
```

### QPOP

---

```
use      : qpop i<id> [Q] <n>
brief    : delete and optionally show next (oldest) n entries from queue
input    : id   - queue ID 0:meas 1:cycle
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 [id]
brief    : print queue summary
input    : id    - queue ID 0:meas 1:cycle
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 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:px[*]bcrhdv]
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  : out - format output (default: PDHV, start=0 c=-1 (all))
        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)
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:[pbcrx[x*]hdv]
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 <file>
brief    : show log record summary
input    : file - log file name (e.g. dat2003.000)
return   : summary
```

## LSEQ

---

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

## 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|H0]  
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 expansion escape character support.  
      fmt is a quoted format string, which may include , escaped variables (with \$ prefix), printf formats

```
var are variables (without $ prefix)
println variant automatically appends CRLF to string
return : formatted console message (upper case)
examples :
  >>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

---

```
enable using WITH_UI_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
        [deprecated; use PRINT, PRINTLN is recommended]
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) [1]
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 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/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
```

## 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 [TEST]

---

### 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]

```

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

examples :

```

# clear all IO bits (except for trancieivers)
clr bits p* v* a* 1
# set ADC bits
set bits a*
# store current IO state in CX0
# 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
:pass
# test passed
println "set bits a* [OK]"
:cont

```

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 (!dir, ms)
input    : cycles   - pulse duration (ms)
input    : id       - bit number or name (see SET BITS)

```

## Group [LI] (licor)

---

## 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 concentration]

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   : none
```

---

## 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 [SCR] (script)

---

---

### 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 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 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 [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.

## MDIO

---

```
use      : mdio [ vd<vdir> vt<vton> ph<pton> pl<ptoff> pn<cycles>
                 <vset> <pump> +sea -sea +air -air +li -li +v -v <preset> d<delay> show]

brief    : Assert digital IO and macros w/ optional delay between operations.
           Operations are performed left to right.
```

#### General:

```
input : delay - delay time (msec)
input : +/-v - verbose output on/off
input : show - show current settings, defaults
```

#### Manual Pump/Valve Control [1]:

```
input : vdir - valve direction:H|L
input : vton - valve pulse time (msec)
input : pton - pump pulse on time (msec)
input : ptoff - pump pulse off time (msec)
input : cycles - pump cycles
```

#### Macro Control [1]:

```
input : mt - macro (preset) valve pulse time (msec)
input : mh,ml - macro (preset) pump pulse time (msec)
input : mn - macro (preset) pump cycles
```

#### Operation Presets [2,3,4]:

```
input : +/-sea - seawater pump on/off
input : +/-air - air pump on/off
input : +/-li - licor on/off
```

```
input : <vset> - switch valve set :
```

```
iofill ioair iopurge iozero ioszero iostd iostdfill iomeas iorest
```

```
input : <preset> - macro presets :
```

```
fill air purge zero szero std stdfill sample mterm rest
```

```
input : <pump> - cycle pump :
```

```
psample pacid
```

#### Examples

##### PURGE:

```
# Do IO setup (set up valve IO, w/ pump, delays, etc.)
mdio purge
# Set up valve IO only
mdio iopurge
```

##### MEAS:

```
# Do IO setup (set up valve IO, w/ pump, delays, etc.)
mdio sample
# Take measurements
mmeas preb sample postb
# Turn off air pump and licor
mdio mterm
```

[1] vt, 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)
- vt, ph, pl, pn do not apply to macro presets
- mt, mh, ml, mn apply only to macro presets

[2] mterm turns off AIR\_PUMP and LICOR after sampling

[3] pn and pump bits (pacid and/or psample) must be set to activate pump  
pump bits accumulate (are OR'd) until pn specified (e.g. mdio pacid psample pn3 )  
pn may be set before or after pacid/psample  
pn, mn are sticky (per command line), may set multiple times  
pump bits (pacid, psample) reset on execution

[4] Valve preset (vset) definitions

```
iofill (SV1A|SV2A|SV3A|SV5A|SV6A)
ioair (SV1A|SV2A|SV3A|SV5A|SV6A)
```

```

iopurge  (SV1B|SV2B|SV3A|SV5A|SV6A)
iozero   (SV1A|SV2A|SV3B|SV5B|SV6A)
zvent    (SV5B)
ioszero  (SV1A|SV2A|SV3B|SV5B|SV6A)
ioszvent (SV5A)
iostd    (SV1A|SV5A|SV6A)
iostdfill (SV4A)
iomeas   (SV2A|SV3A|SV5B|SV6A)
ioreset  (SV1A|SV2A|SV3A|SV5A|SV6A)

```

## MMEAS

---

```

use      : mmeas [ <rtype_id> <rmeas_id> <mmeas_id> p<period> b<blank>
                  <mtime> n<mcount> w<cal_wr> d<delay> meas <preset> mbreset
                  stat statv out<src>[:<output_ctl>] ]

brief    : licor measurement cycle
options:
rtype_id - record type : RTCYCLE RTMEAS RTX<x>

rmeas_id - record measurement ID : RMSAMPLE RMCAL RMX<x>

mmeas_id - measurement ID : MMAIR  MMSAMPLE MMX<x> MMPRES MMSPAN MMPOSTS
          MMPREZ MMZERO  MMPOSTZ MMPREB MMBLINE MMPOSTB

mtime    - measurement time (msec)
mcount   - measurement count (alternative to mtime)
period   - licor sample period (sec >=0.5)
blank    - blank period (msec >=0)
cal_wr   - write calibration to licor (+|-))
delay    - delay (msec)
mbreset  - reset measurement buffer
meas     - take measurement using current settings
preset   - pre-configured settings: zero air std sample prebl postbl zcal scal

stat      - show measurement buffer summary

output_ctl - output control: outm|outr|out<src>[:<dest>[,<fmt>]]

          src - source [M,R,<n>]
              M:last measurement R:record <n>:(n+1)th measurement (zero-indexed)

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

          fmt - format flags [PBRCXX] (optional, console only)
                 P: Pretty B:Base85 C:CSV R:raw
                 X:hex XX: PrettyHex X*: hex+PrettyHex

Example:
mmeas outm          output last meas to console (Pretty)
mmeas outm:c,b      output last meas to console (Base85)
mmeas outm:c,pxx    output last meas to console (Pretty & PrettyHex)
mmeas outr:q        output record to (cycle) queue
mmeas outm:q        output measurement to (meas) queue
mmeas out4:q        output 5th (i.e. index=4) measurement to (meas) queue

```

## Group [REF] (reference)

---

### 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 |
| LICOR_DC   | 22  | 0x00400000 | 0000 0000 0100 0000 0000 0000 0000 0000 |
| ADC0       | 23  | 0x00800000 | 0000 0000 1000 0000 0000 0000 0000 0000 |
| ADC1       | 24  | 0x01000000 | 0000 0001 0000 0000 0000 0000 0000 0000 |
| ADC2       | 25  | 0x02000000 | 0000 0010 0000 0000 0000 0000 0000 0000 |
| XCVR_HOST  | 26  | 0x04000000 | 0000 0100 0000 0000 0000 0000 0000 0000 |
| XCVR_LICOR | 27  | 0x08000000 | 0000 1000 0000 0000 0000 0000 0000 0000 |
| XCVR_CONS  | 28  | 0x10000000 | 0001 0000 0000 0000 0000 0000 0000 0000 |