

MBARI Carbon WG Console Help

Contents

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

Entries in each topic are arranged alphabetically.

General syntax guidelines

The guiding principles for syntax and semantic design in the user interface (UI) include:

- simplicity (easy to type and understand)
- balance minimal command length with readability
- structural and behavioral consistency
- one good way to do anything : full, fine-grained functional exposure with minimal redundancy

To that end, these conventions are used:

- `<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 e.g. `[ <foo> <bar> ]` are optional but may contain required elements. For example `[op <var>]` indicates that 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 e.g., `[<foo>...]` means one or more `<foo>` 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`

- A colon is used to delimit multi-valued options (i.e. a list), which are comma-delimited e.g. `out:cr,pdh`
- If an option list contains multi-valued variables (sub-list), the sub-list is delimited using `'/'`, e.g. `sh1:v1a/v3,5000`
- `*` is a wildcard, signifying all/any

Topic APP_EX

(pCO₂/DIC experiment)

Application specific : Measurement control macros tailored to control pCO₂ and DIC measurements using LI-COR 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 LI-COR 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 initialize the operation profile, apply optional overrides, and/or inspect the configuration before performing the measurement.

The TS (take sample) option automatically 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 operations.  
          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. pulse)  
use: phl:<iobits,ton,toff,cycles,[post-delay]>
```

```
SHL, SLH  - pulse IO bit(s) HI/LO, LO/HI (single cycle, toff=0 e.g. set)  
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:

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

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

+/-AIR - air pump on/off
 +/-SEA - seawater pump on/off
 +/-LI - LI-COR 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 LI-COR
REST    - Put valves in rest state
VEX     - Exercise valves

```

notes :

- [1] Operations are evaluated and executed left to right.
- [2] Command and options are case-insensitive.
- [3] Command length limited by UI_CMD_BUF_BYTES (console context) or SCR_LINE_BUFFER_BYTES (in script context). Use multiple lines for long sequences.
- [4] 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)
- [5] 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

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 LI-COR
# 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 pump pulse
# - put IO in rest configuration
mdio shl:v1a/v3,5000 phl:pa,300,200,15 +air d10000 -air pn10 spu

```

MMEAS

```

use      : mmeas <cmd>:[options]

brief    : Configure and/or execute one or more LI-COR 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, record type 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/disables LI-COR 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 LI-COR period to 1.0
 usr2:p1.0

LOAD,TM - Load measurement profile(s), optionally apply overrides
 LOAD will load and configure a profile, but requires execution
 This enables load/inspect without running.
 TM automatically runs specified profile(s) immediately
 use: LOAD:[option,...] or TM:[option,...]

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 LI-COR samples
 mmeas meas:sample,n14,run
 mmeas tm:sample,b2000,air

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

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

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

where

[CHLQ] - destination flags

C : console
H : host
L : log
Q : queue

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

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 (Base
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 LI-COR sample period (decimal sec, 0.0 or >=0.1)
                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 instrument
                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

```

notes:

- [1] Operations are evaluated and executed left to right.
- [2] Command and options are case-insensitive.
- [3] Command length limited by UI_CMD_BUF_BYTES (console context)
or SCR_LINE_BUFFER_BYTES (in script context). Use multiple lines.

```
        for long sequences.

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 c
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
```

Contents

Topic APP_LI

(LI-COR)

Application specific : LI-COR 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)
return     : none
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]

```

LI INIT

```

use      : li init
brief    : initialize LI-COR
return   : none
examples :
    # initialize LI-COR
    li init

```

LI RATE

```

use      : li rate <period>
brief    : set LI-COR output rate [1]
input    : period - output period (s) [range 0.0 or >=0.5]
return   : none

notes    :
    [1] The use of OR instead of RATE is also supported

examples :
    # set LI-COR output rate to 0.5s
    li rate 0.5

```

LI STAT

```

use      : li stat [i<indent> v] <tag>...
brief    : output LI-COR 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

```

Contents

Topic LOG

(logging)

Access log entries and metadata.

LOG

```

use        : log <msg>
brief      : add annotation to syslog
input      : msg - message to log
return     : none
examples   :
  # add annotation to syslog
    log 'nice weather we're having'

```

LSEG

```

use        : lseg <hnd> [<val>]
brief      : get/set log segment size [1]
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
notes      :
  [1] Chances are you don't need to change this. Use caution if you c

```

segment size too large can degrade real-time performance (file interfering with sample buffering and timing).

LSHOW

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

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
examples :
    # show last three records if dat2010.000 pretty format, include d
    lcat dat2010.000 out:pdh
```

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<isol806>
               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

notes    :

```

```
[1] 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 (useful for machine-machine queries)
        e - epoch time (posix, CSV only)
        i - ISO8601 time (CSV only)
        h - include CSV header (CSV only)

examples :
    # show data log summary (pretty and CSV, ISO8601 time format, inc
    linfo dat2010.000 out:pcih
```

LSEQ

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

Contents

Topic QUEUE

(sample queue management)

The system exposes 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 queue contents.

This can be useful for an operation model in which a client periodically

requests the latest measurements or records. The queue API may also be used to transfer measurements or records to the data log.

The MMEAS API uses an internal cycle measurement queue to store measurements. The cycle is not directly accessible to users. MMEAS output commands may be used to transfer cycle queue contents to the measurement and record queues or data log.

QCOUNT

```
use      : qcount i<id>
brief    : return number of queue entries
input    : id      - queue ID 0:meas 1:cycle
return   : number of queue entries
examples :
    # show number of entries in measurement queue
    qcount i0
```

QCLEAR

```
use      : qclear i<id>
brief    : delete all queue entries
input    : id      - queue ID 0:meas 1:cycle
return   : none
examples :
    # empty the cycle queue
    qclear i1
```

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
examples :
    # inspect the first 2 entries in the measurement queue
    qpeek i0 0 2

```

QPOP

```

use      : qpop i<id> [Q] [OUT:<dest>[,<fmt>] <n>
brief    : delete and optionally show next (oldest) n entries from
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)
examples :
    # remove the first 3 entries from the measurement queue,
    # send contents to console in Base85 format
    qpeek i0 2 out:c,bdh

```

QSHOW

```

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

```

Contents

Topic 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
examples :
    # enable script debugging for script ID 1
    scr sdebug + 1
```

SCR DELAY

```
use      : scr delay <time_ms> <id>
brief    : delay specified period before executing next command (su:
           Run timer continues to run and resumes when delay expires
input    : time_ms - delay time (ms)
input    : id      - script name or handle
return   : none
examples :
    # pause scr 0 for 30 sec
    scr delay 30000 0
```

SCR END

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

SCR JOG

```
use      : scr jog <id>
brief    : enter manual mode and begin single-step execution
          : subsequent calls advance the specified script one line
input    : id - script name or handle
return   : none
examples :
  # put scr 2 in manual mode, advance one step
  scr jog 2
  # advance one step
  scr jog 2
  # resume (continue running from current line)
  scr run 2
```

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 (m
                overwrite is enabled by default.
return   : confirmation or error message
examples :
  # load "3step.scr" (fail if already loaded)
  scr load 3step.scr
  # load a.scr and b.scr, overwrite b.SCR
  scr load a.scr o+ b.scr
```

SCR RUN

```
use      : scr run <id>
brief    : run the specified script [1]
input    : id - script name or handle
return   : none

notes    :
  [1] Script lines run to completion without interruption.
      Console and Host IO are processed between script lines, allowing
      scripts to be interrupted, e.g. Response to user IO may be slow.
  [2] Scripts may be run concurrently, executed one line per cycle,
      starting with script ID 0.
```

```
examples :
# load and run script a.scr
scr load a.scr
scr run a.scr
```

SCR KILL

```
use      : scr kill <id>
brief    : stop running the specified script
           Stops execution and cancels schedule.
           Does not change hardware states.
input    : id - script name or handle
return   : none
examples :
# stop script a.scr
scr kill a.scr
# stop script ID 0
scr kill 0
```

SCR RESET

```
use      : scr reset <id>
brief    : reset (cancels) the delay timer for specified script
           Does not change the run timer, i.e. run timer continues running
           during delay, and the resumes when delay timer is reset.
           Use sync to reschedule or kill to stop.
input    : id - script name or handle
return   : none
examples :
# reset script a.scr delay timer
scr reset a.scr
```

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
```

```
examples :
# run a.scr every 20 min, start now
scr sch 1200000 y a.scr
# set b.scr schedule every 5 min, but don't start
scr sch 3000000 b.scr
# run b.scr once (even if schedule set)
# start b.scr using schedule
scr sync b.scr
```

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
examples :
# show script status/summary for all scripts
scr show
# show script status/summary for script ID 0
scr show 0
# show script status/summary for script a.scr
scr show a.scr
```

SCR SYNC

```
use      : scr sync <time_ms> <id>
brief    : start script after optional delay
           Shifts scheduled start time.
input    : time_ms - delay time (ms) (default is 0)
return   : none
examples :
# set script schedule to run every 60 min
scr sch 3600000 n a.scr
# start in 5 minutes
scr sync 300000 a.scr
```

SCR UNLOAD

```
use      : scr unload <id>
brief    : unload specified script.
input    : id - script name or handle
```

```
return : none
examples :
  # unload a.scr
    scr unload a.scr
  # unload script ID 0
    scr unload 0
```

Contents

Topic 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 [0:2]
output   : DXn - store result in DXn [0 < n < SCR_DX_COUNT-1] (optional)
return   : ADC - voltage [0-5V]
examples :
  # read ADC0 into DX0 and ADC2 into DX1
    get ad 0:dx0 2:dx1
```

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

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

```

examples :
  # show bit states
  get bits
  # show bit states, read into CX0
  get bits CX0
  # show stored bits in cx0 (decimal)
  creg CX0
  # show stored bits in cx0 (hex)
  print "%LX\r\n" CX0

```

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
examples :
  # get next computed time
  get next

```

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
examples :
  # show debug and verbose setting
  get print

```

GET SER

```

use      : get ser [N<m>] <port>
brief    : read from serial port string until
          - m bytes received (m>0)
          - no bytes available (m==0)
          - indefinitely (m<0)

          May interrupt using CTRL-D any time.
          Sends received bytes to console.

input    : port - serial port [0:2|C|H|L]

```

```
0,H : host
1,L : LI-COR
2,C : console

return : bytes read
examples :
    # get 50 bytes input from the LI-COR port
    get ser L N50
```

GET TIME

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

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
examples :
    # show all configuration variables
    get var
    # show configuration variable ECHO
    get var echo
```

[Contents](#)

Topic SET/CLR

(change system hardware state)

Utilities for changing system hardware state parameters.
The set API provides 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
examples :
  # set host port communications rate to 19200
  set baud H 19200
```

SET BITS

```
use      : set bits [<id>...]
brief    : set (HI) one or more digital IO bits
          see also clr bits, pulse bits, and MDIO

input    : id - bit number or name
          #<n> : bit n (decimal)
              0:19 valves/acid pump/sample pump
              20:21 pump
              22:22 LI-COR
              23:24 analog
              26:28 RS232 drivers
          x<n> : bit mask n [0-FFFFFF] (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    : LI-COR DC power
```

```

uh      : host UART xcvr
ul      : LI-COR UART xcvr
uc      : console UART xcvr

return  : none
examples :
# set air pump, valves SV3A and SV7A+B and the licor power and tra
set bits pr v3a v7 ul 1

```

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
examples :
# clear all pump, valve and ADC enable bits
set bits p* v* a*

```

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 supports multiple levels (0-3)
return   : none
examples :
# enable debug printing and verbose level 2
set print dl v2

```

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
examples :
# send "hello\r\n" to the host port

```

```
set ser H "hello\r\n"
```

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   [00:23]
input    : mm      - minute [00:59]
input    : ss      - second [00:59]
return   : new time
examples :
# set time to 2010/10/03T11:13:42
set time 2020 10 3 11 13 42
```

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
examples :
# set var FOO_DBL (double)
set var FOO_DBL 3.14
```

[Contents](#)

Topic 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 (single cycle, e.g. to
input    : dir      - direction [L|0|H|1]
input    : delay_ms - pulse duration (ms)
input    : id       - bit number or name (see SET BITS)
return   : none
examples :
# pulse valves v1a and v4a,b H for 200 ms then return to L
pulse bits H 200 v1a v4
```

PULSE NBITS

```
use      : pulse nbits <dir> <ta> <tb> <n> bits [<id>...]
brief    : pulse one or more digital IO bits w/ specified duty cycle
          (e.g. to actuate pumps)
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)
return   : none
examples :
# pulse acid pump 10 cycles 200ms H/ 300ms L
pulse bits H 200 300 10 pa
```

Contents

Topic 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
```

```
notes    :
  [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 trancivers)
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
:pass
# test passed
println "set bits a* [OK]"
:cont
```

[Contents](#)

Topic 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 unsigned integer counters
DREG : 8 x 32-bit 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 accidental modification through concurrent use.

CREG

```
use      : creg [<reg>] [<reg><op><val>]...
brief    : show, manipulate counter reg (CX0:SCR_CX_COUNT)
input    : reg - register [CXn|*]
input    : op   - operator [=,--,++,+,-,*,/,%=]
input    : val  - value [signed 32-bit integer]
return   : requested value(s)

notes    :
[1] operators "++" and "--" don't require (or support) values
[2] CREG values are unsigned 32-bit integers

examples :
# operations
creg cx2=0 cx2 cx2+=5 cx2 cx2*=3 cx2%=6 *
# demonstrate rollover
creg cx2=0 cx2 cx2-- cx2 cx2++ cx2
```

DREG

```
use      : dreg [<reg>] [<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)

notes    :
```

```
[1] operators "++" and "--" don't require (or support) values
[2] DREG values are 32-bit doubles
```

examples:

```
# operations
dreg dx2=0 dx2++ dx2 dx2+=2.5 dx2 dx2/=3.1415 *
```

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)
```

notes :

```
[1] operator "!" doesn't require (or support) values
[2] FREG values are boolean
```

examples :

```
# operations
freg fx2 fx2=1 fx2 fx2=0 fx2 fx2! fx2
```

Contents

Topic FSYS

(file system)

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

CAT

```
use      : cat [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> [2,3]
<mode>     : range type [blims][+-]
              b - byte mode
              l - line mode
              i - time mode (ISO8601)           [4]
              m - time mode (epoch msec)       [4]
              s - time mode (epoch sec)        [4]
              - - relative to end of file      [5]
<val>      : range value
              line/byte count (index >=0)
              timestamp (epoch sec/msec or ISO8601)

notes      :
[1] there are two cat implementations, this feature is selected by
[2] range specifiers optional; use empty field (no space) if unspecified
[3] valid end modes include [blims]; others ignored. Start/end time
[4] file lines must start with specified timestamp (leading space ok)
[5] 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,2020-03-28T00:00:00,1,10 sys2000.000
# show first 10 lines of foo.csv >= 1592974762 (Tue Jun 23 21:59:22)
xcat r:s,1592974762,1,10 foo.csv
# show 100 bytes starting at line 10 sys2000.000
xcat r:l,10,b,100 sys2000.000

```

CFGEX

```

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

```

```

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

```

```

examples:

```

```
# save configuration to foo.cfg
cxgex foo.cfg
```

CP|MV

```
use      : cp <src> <dest>
brief    : copy contents from src to dest [1]
input    : src  - file to copy FROM
input    : dest - file to copy TO
return   : none

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

examples :
  # copy foo.cfg to bar.bak
  cp foo.cfg bar.bak
```

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 text file
input    : file - file to evaluate
return   : number of lines, execution time

notes    :
  [1] any console input stops processing
  [2] large files may take a long time ( ~1s/1k lines)

examples :
  # count lines in sys2010.000
  lc sys2010.000
```

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    :
  [1] pager does not support terminal control or page buffering;
       output begins at the bottom of the screen.

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
examples:
  # count lines in sys2010.000
```

REN

```
use      : ren <from> <to>
brief    : change file name (preserves existing file)
input    : from - current file name
input    : to   - new file name
return   : none
examples :
  # rename junk.txt foo.txt
  ren junk.txt foo.txt
```

RM

```
use      : rm <file> [<file>...]
brief    : delete one or more files
          - does not prompt for confirmation
          - cannot be undone
input    : file - file to delete
return   : none
examples :
# delete foo.txt
rm foo.txt
```

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]
return   :
notes    :
[1] not for use with binary files
[2] dest will be overwritten if it exists
    otherwise, it will be created
    upload times out after 4s, or may be terminated using CTRL-D

[3] 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

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

examples :
# upload file to test.scr
upload test.scr
[initiate file upload in terminal, e.g. CTRL-A Y in minicom]
[wait for timeout or press CTRL-D when finished]
```

Contents

Topic DIAG

(diagnostics/debug)

Diagnostics and system utilities

HELP|?

```
use      : help [V]
brief    : show help info (same as less help.md)
input    : V - verbose output (this file) (optional)
return   : help information
examples :
  # show help file
  help v
  # show command listing
  help
```

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 s

notes    :
  [1] up/down arrow keys may be used to traverse the queue

examples :
  # show history queue
  hist
```

CON

```
use      : con <port>
brief    : open console to device
          [terminate session using CTRL-D]
input    : port - serial port
          L: LI-COR
```

```

        H: Host port
return   : none
examples :
    # connect to LI-COR
    con li
    [do stuff]
    [type CTRL-D to exit]

```

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 // LI-COR
          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  :
    # disable debug output
    debug =0
    SYS_MDFLAGS=x00000000
    # enable scr module debug output
    debug +4
    SYS_MDFLAGS=x00000004
    # enable LI-COR module debug output
    debug +8
    SYS_MDFLAGS=x0000000c
    # disable LI-COR module debug output
    debug -8
    SYS_MDFLAGS=x00000004
    # show debug flags
    debug
    SYS_MDFLAGS=x00000004

```

PMEM

```

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

```

```
examples :  
# show memory pool  
pmem
```

PRINT, PRINTLN

```
use      : print <fmt> [<var>...]  
brief    : print formatted messages to console using variable expansion  
           character support. Useful for script testing/debugging.  
  
           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  
  
# print CX using alternative (decimal) format (automatically terminates)  
println "CX0:%ld" CX0  
CX0:20
```

VER

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

UPTIME

```
use      : uptime
brief    : return elapsed time since last power cycle
return   : time since last power cycle
examples :
    # show system up time
    uptime
```

RESET

```
use      : reboot
brief    : reboot the firmware
return   : none
examples :
    # reset system
    reset
```

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, $
           escape characters are supported in quoted strings [\r\n\t]
return   : 'hello' [args...]
examples :
    # print hello message
    hello foo $CX0
```

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)
```

```
notes      :
  [1] **IMPORTANT** : for hardware testing; disconnect user equipment

examples :
  # run sweep with 0.5 sec delay/switch
  sweep 500
```

Contents

Topic Beta

(deprecated and/or beta features)

These features include

CAT

(alternative implementation, not enabled, see notes)

```
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
notes    :
  [1] there are two cat implementations, this feature is selected by
```

ECHO

(not enabled by default, use PRINT, PRINTLN)

```
use      : echo [<op><dest>] [<msg>...]
brief    : print messages using variable expansion and escape characters
```

```
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. $v)
                  escape characters are supported in quoted strings
return     : expanded message
notes      :
    [1] this feature is selected by enabling WITH_UI_ECHO
```

PROTO

(code prototyping feature)

```
use        : proto (args)
brief      : runs prototype code features [for developers]
input      : args - arguments (varies)
return     : varies
notes      :
    [1] this is intended for use by developers; use caution and discon
        before using.
    [2] behavior depends on compile-time feature selection. Existing p
        configured in cwgsys.h
    [3] see proto.c, proto.h for integration examples
```

[Contents](#)

Topic REF

(reference)

Reference information

Configuration Variables

Key	Type	Value	Description
LOG_RAW	bool	1	log raw LI-COR records
LICOR_MODEL	i16	850	LI-COR model/protocol
SERIAL_N	u16	XC024	hardware serial number
LI_SPANC	dbl	3.14	LI-COR SPAN cal value
Unused			
VERBOSE	bool	0	
ECHO	bool	1	
LI_CFLAGS	u32	X000003FF	
LI_OFLAGS	u32	X001F7FFF	
VALVE_TYPE	u16	X004D	
AUTO_OUT	bool	0	
FOO*	-	-	

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

Contents