What's new

Previously…

A number of different structures were used to represent data, differentiated loosely by measurement context. **Data were stored using several different types of ASCII CSV records**, each with unique (and in some cases variable) content.

The record types were measurement-centric, representing a portion of a sample or calibration cycle.

However…

Although the records contained some information to help identify their content and measurement context, **post-processing was made more difficult by ambiguous record typing**.

The different **record structures were not intuitive** to understand. **Multiple data handlers** were used to process raw records for different measurement types.

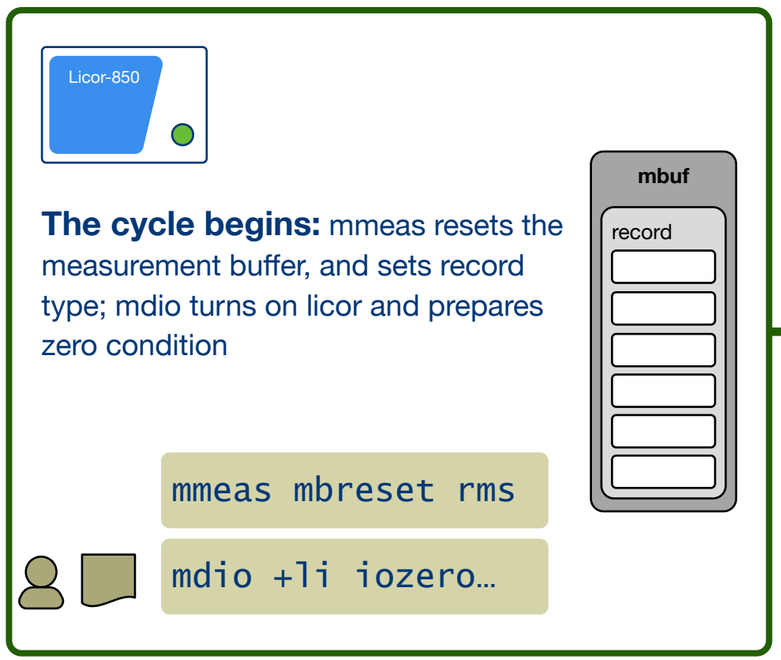
Also, although it is convenient, **ASCII is not efficient** or useful for internal manipulation.

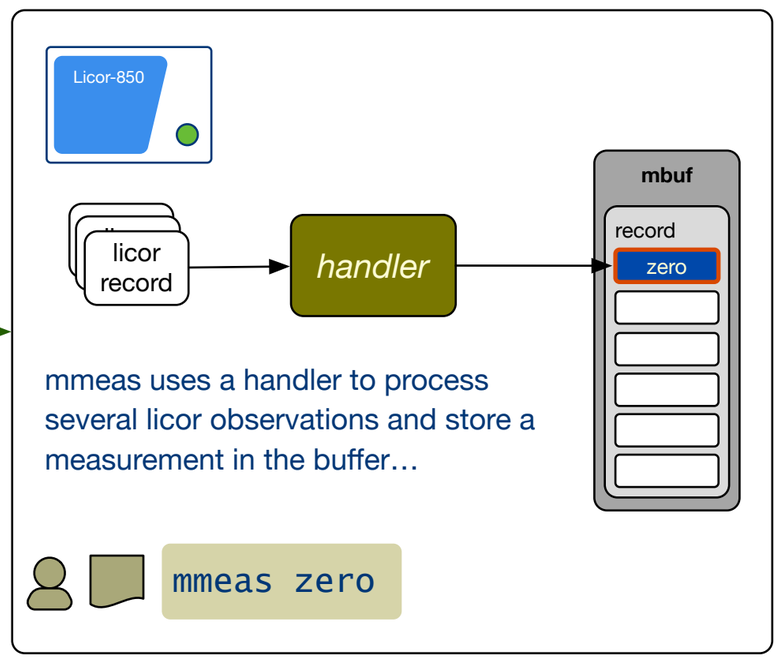
And so…

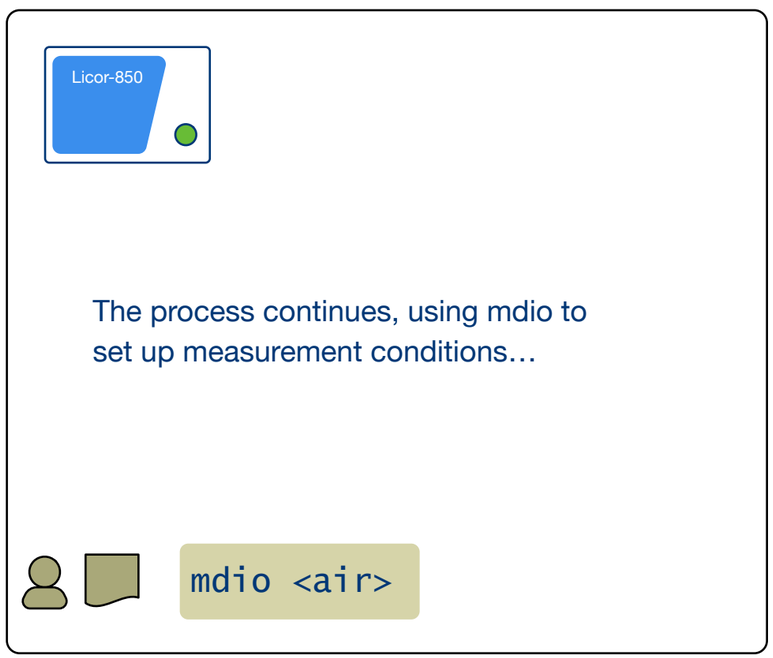
To address these issues, the internal data handling structures have been revised, making them more cycle-centric, enabling **increased data yield, more efficient storage, and simplified post-processing workflows**. Major changes include:

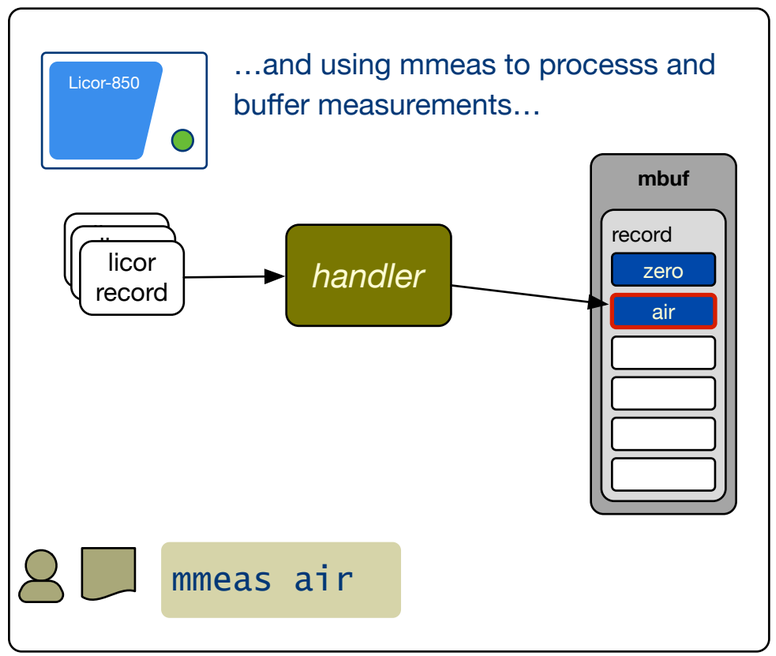
- ***Binary record log***
- ***Single, consistent variable-length record format***
- ***Unambiguous measurement context definition***
- ***An improved output API enables access to records and measurements in a variety of formats****, including human-readable summaries and modem transfer encodings*
- *Relatively minor changes to the mmeas and queue command line semantics*
- *Deprecated code and commands: li tm, li nmeas*

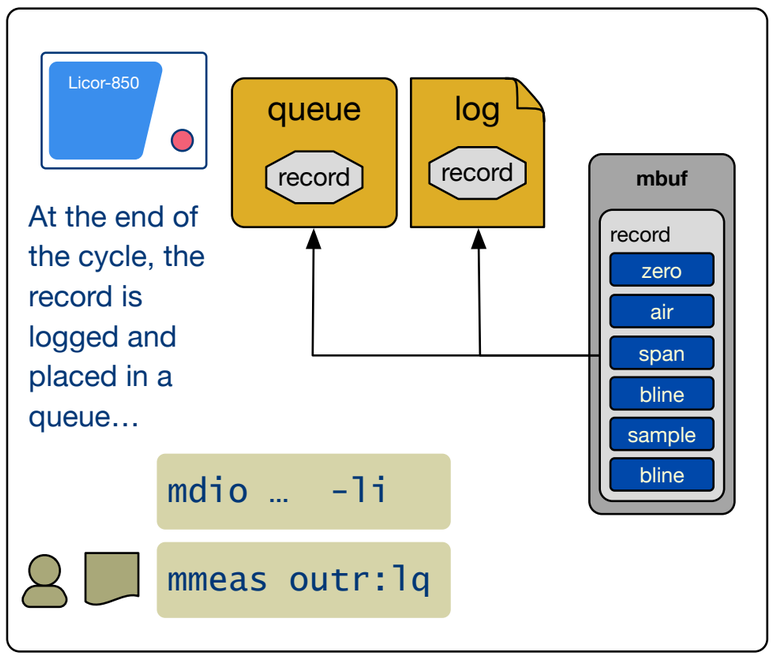
Sampling sequence

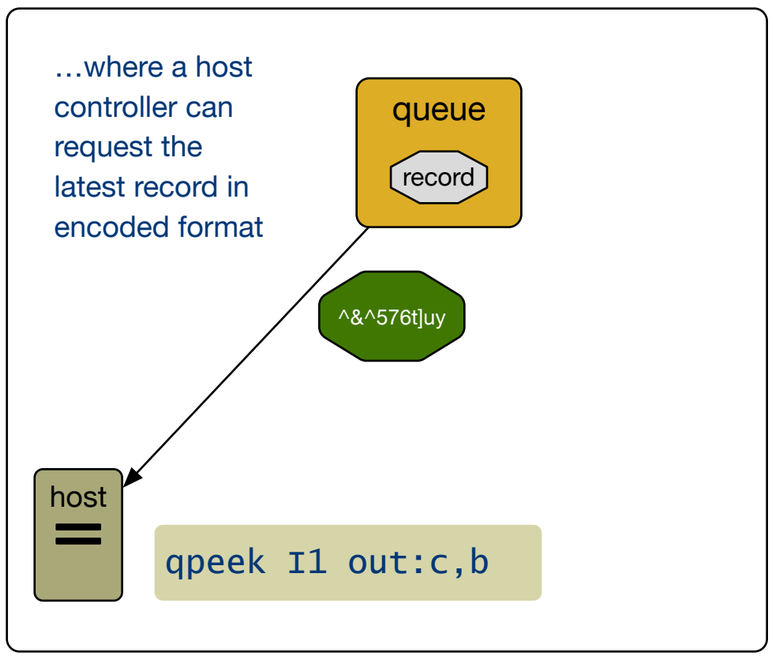












Output formats

The contents of the cycle buffer and queues may be inspected using several output formats.

Output formats are provided as arguments to mmeas and the queue commands. Commands syntax for displaying the latest measurement on the console are in various formats are shown here.

qpeek and qpop use similar syntax (“out:” instead of “out<src>:”). Output is sent to the console using Pretty format by default.

```
qpeek I0 out:c,xx
```

Human-readable formats
(partial output shown)

Pretty - Formatted table

```
mmeas outm:c,p
```

```
            self           1238
            sync       0CEA4C02
       timestamp  1582838434000
         type_id           0052
         meas_id           0061
           seq_n              0
             len            280
      meas_count              1
     measurement              0
            self           1254
    timestamp[0]  1582842029500
    timestamp[1]  1582842034000
         meas_id           0001
          stat_n             10
     min/max/M/S
             co2 +1.064e+01 +1.090e+01 +1.074e+01 +7.230e-02
          co2abs +1.155e-02 +1.182e-02 +1.166e-02 +7.737e-05
             h2o +8.784e+00 +8.786e+00 +8.785e+00 +5.663e-04
          h2oabs +5.345e+00 +5.347e+00 +5.346e+00 +8.060e-04
       …
```

PrettyHex - Formatted hexidecimal

```
mmeas outm:c,xx
```

```
00000000: EF BE AD DE 02 00 00 00 F8 54 36 88 70 01 00 00
00000010: 00 00 00 00 01 00 FC 00 00 00 2A 08 E4 31 6D 88
00000020: 70 01 00 00 78 43 6D 88 70 01 00 00 00 00 0A 00
00000030: A4 C0 E9 40 A3 E1 00 3C F6 B2 18 41 4F 91 D1 40
00000040: 6D 14 9E 3D B8 8C CF 41 E7 5A CC 42 9D 7C 31 41
00000050: 00 00 00 00 A2 FA 84 4A 10 B8 93 4A EB 6C 01 4A
…
```

Machine-Machine formats
(partial output shown)

Base85 - Efficient ASCII encoding

```
mmeas outm:c,b
```

```
[4L{X0SSi2{/6ecz#/}s000000rr@q00$h<s$Icz#/…
```

Hex - Unformatted hexidecimal

```
mmeas outm:c,x
```

```
EFBDDE02000000B83B3588700100000000000000…
```

CSV - Comma-delimited ASCII

```
mmeas outm:c,c
```

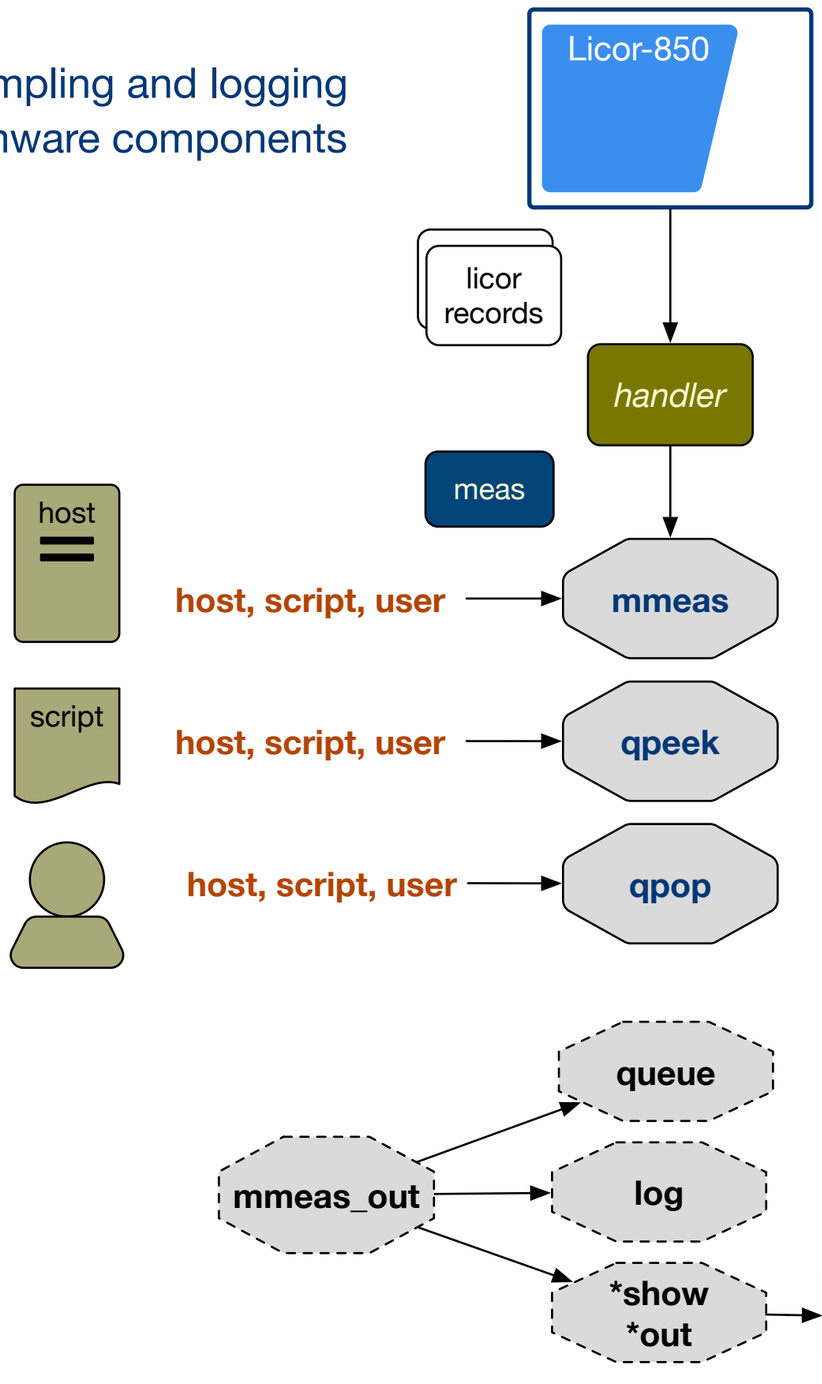
```
1582830433000,0042,004D,0,1512,6,…
```

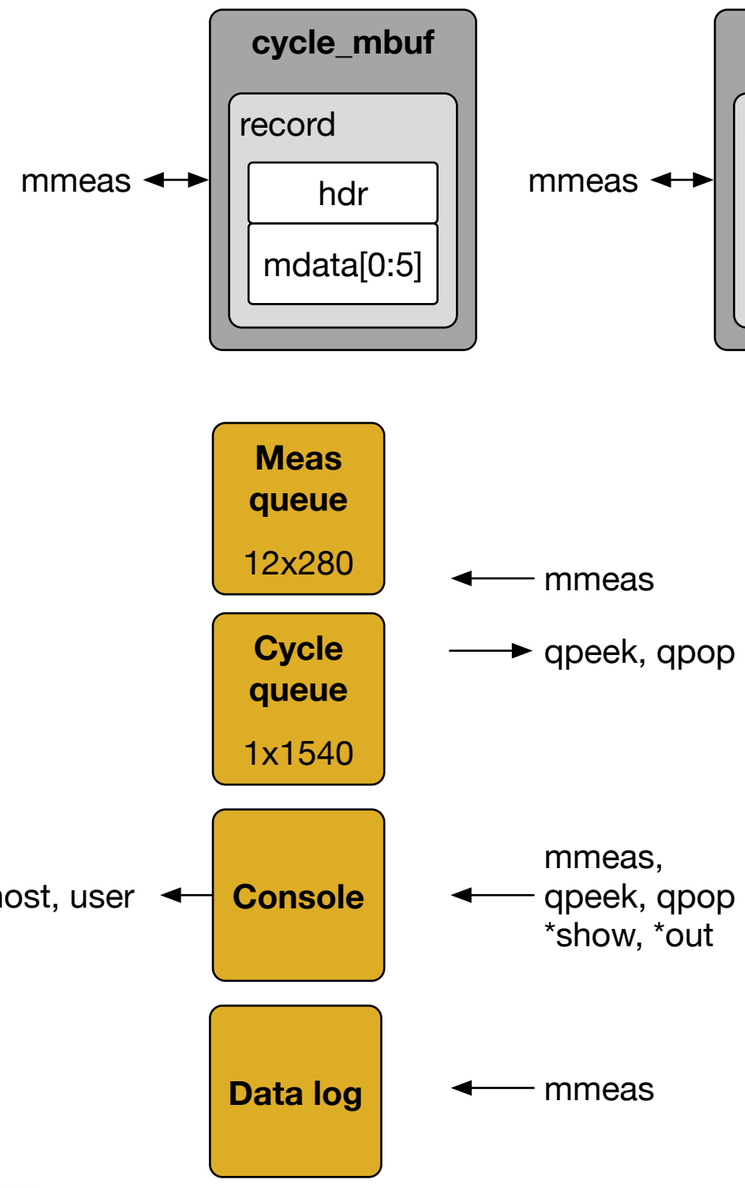
Raw - Just the bytes

```
mmeas outm:c,r
```

Data collection architecture

Sampling and logging firmware components

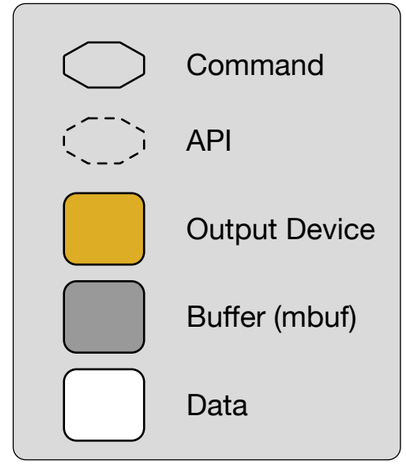




The transfer buffer is used by mmeas to format measurement data as records for placement in the measurement queue.

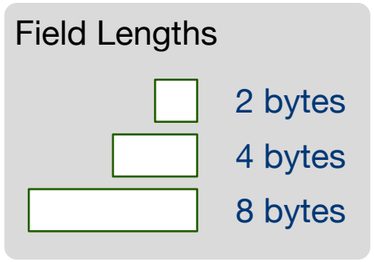
Measurement queue holds records consisting of a single measurement. Typically used for test/debug, but may be logged.

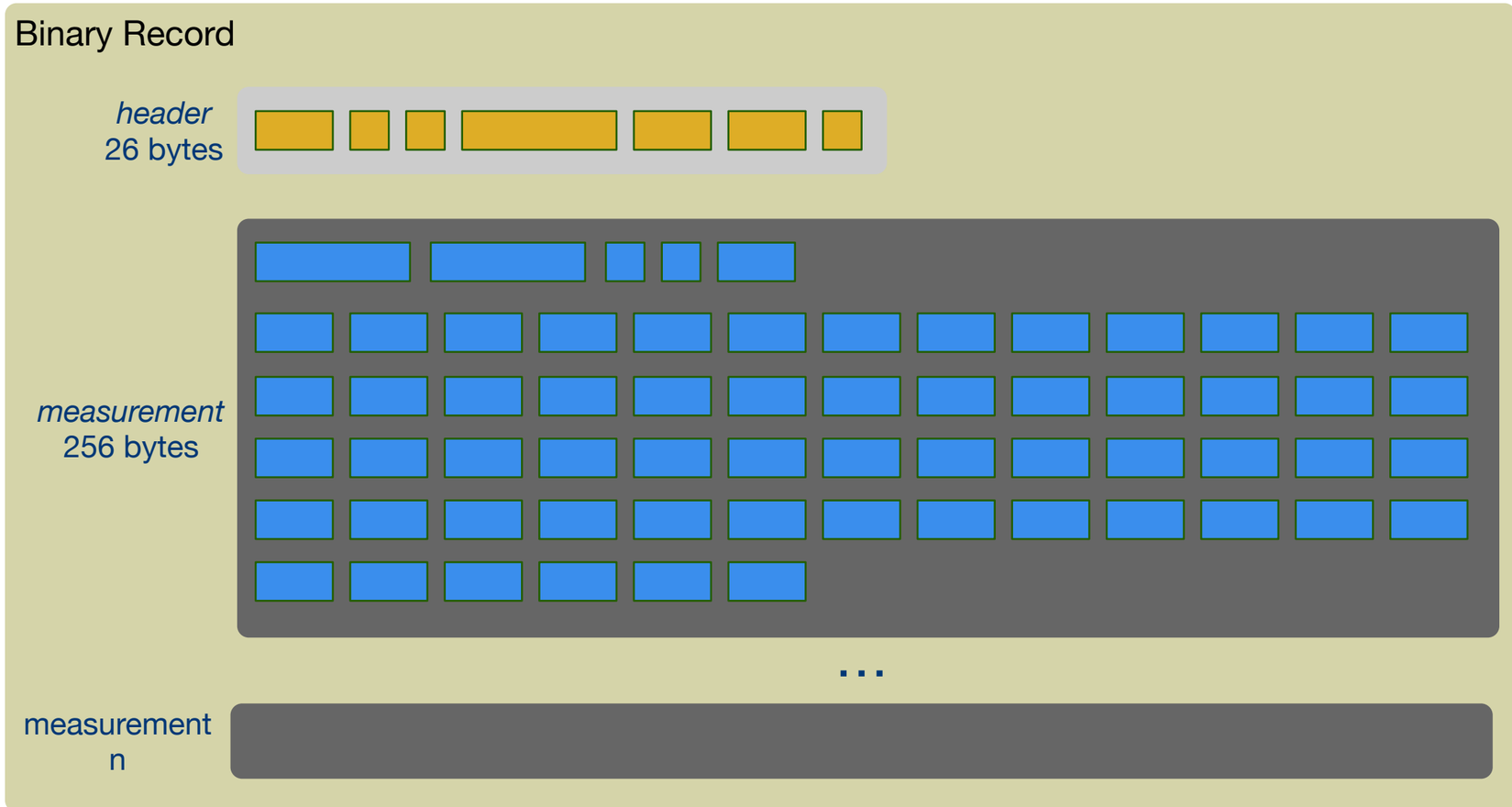
Cycle queue holds latest record, consisting of a multiple measurements (typically logged, e.g. sample, cal cycles)

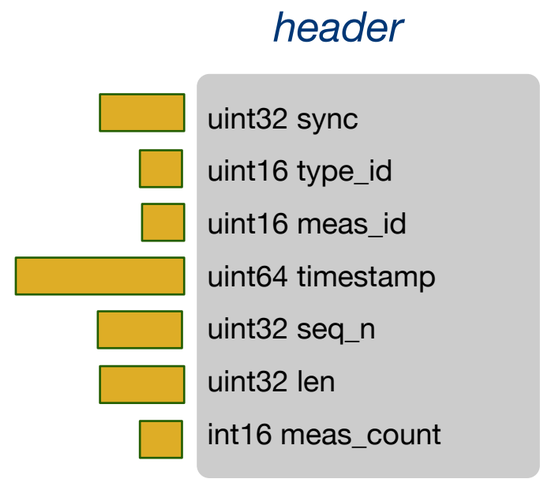


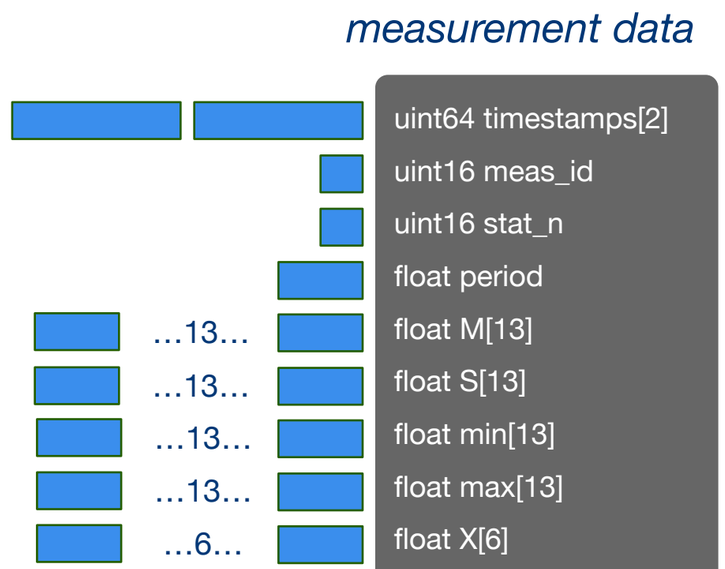
Binary record map

Each binary record consists of a header and n (typically up to 6) measurements, indicated by header meas-count field.









Record header

Binary record descriptor

Header

Describes variable length record

sync

4-byte pattern marks start of record; used for stream recovery

type_id

Indicates record structure/format

meas_id

Indicates general record context for included measurements (e.g. sample, cal, etc.

timestamp

Time record recorded

seq_n

Record sequence number

len

Record size (bytes, not including header)

meas_count

Number of measurements in record

Measurement data

Primary measurement data structure and contents

timestamps

measurement start and end times

meas_id

specifies measurement condition, e.g. baseline, zero, span

stat_n

number of observations used for computation

period

licor output period (s) for this measurement

Measurement Data

One or more processed licor records.
All measurements use this format

- timestamps[2]
- meas_id
- stat_n
- period
- M[13]
- S[13]
- min[13]
- max[13]
- X[6]

CO2 trend line parameters may be calculated from mdata

slope	CO2 trendline slope
offset	CO2 trendline offset
R^2	CO2 trendline correlation

min, max, M, S

Licor parameter minimum, maximum, mean (M) and standard deviation (S) values

co2	co2 concentration
co2abs	absolute CO2
h2o	H2O concentration
h2odewpoi	dewpoint
nt	absolute H2O
h2oabs	measurement cell
celltemp	temperature
cellpres	measurement cell pressure
ivolt	voltage
flowrate	gas flow rate
rawco2	CO2 counts
rawco2ref	CO2 reference counts
rawh2o	H2O counts
rawh2oref	H2O reference counts

X

Calculated parameters

co2_area	area under CO2 curve
co2_ep0	initial CO2 value
co2_ep1	final CO2 value
MX	x-axis (time) mean
SX	x-axis (time) standard deviation
SXY	CO2 variance

Data structures

Firmware record and processing data structures

mbuf structures carry processing state as well as measurement data. They are passed with raw licor records (observations) to handlers, which transform raw data into measurements

Measurement buffer

Computational state and measurement buffer

z_meas_buffer_t

int meas_idx
uint16_t depth
sdata_Z *sdata
z_log_rec record

cycle_mbuf

record
hdr
mdata[0]
mdata[1]
mdata[2]
mdata[3]
mdata[4]
mdata[5]

Log record

One or more related measurements

z_log_rec_t

log_rec_hdr hdr

log_rec_header_t

Record header

Record metadata

uint32 sync
uint16 type_id
uint16 meas_id
uint64 timestamp
uint32 seq_n
uint32 len
int16_t meas_count

mdata_Z_t

Measurement data

One or more processed observation(s):
- min, max, mean, sdev
- derived parameters, e.g. area, trendline

uint64_t timestamps[2]
uint16 meas_id
uint16_t stat_n
float period
double M[ZID_LICOR_FIELDS]
double S[ZID_LICOR_FIELDS]
double min[ZID_LICOR_FIELDS]
double max[ZID_LICOR_FIELDS]
double X[ZID_EXTRA_FIELDS]

sdata_Z_t

State data

Computional state data, not logged

uint64_t timestamp_prev
double M_prev[ZID_LICOR_FIELDS]
double V_prev[ZID_LICOR_FIELDS]
double S_prev[ZID_LICOR_FIELDS]
double MX_prev
double SX_prev
double SXY_prev

licor_orecord_t

Observation record

Raw licor record field values, parsed from XML

uint64_t timestamp
double *fields[FID_LICOR_FIELDS]

	Size (bytes)
header	26
mdata	256
record	26+n*256
full record	1562

Record types

Constants used to define record and measurement types

Record-related

Record Type

Indicates **physical record structure**
[currently only one type]

record_type_id

RTID_ZRECv0	0	Z-record format (variable len)
<other>	-	user-specified (16-bit)

Record Measurement ID

Indicates record context/content

record_meas_id

RMID_NONE	0	unspecified
RMID_SAMPLE	1	sample cycle
RMID_ZCAL	2	zero calibration cycle
RMID_SCAL	3	span calibration cycle
<other>	-	user-specified (16-bit)

Measurement-related

meas_meas_id

MMID_NONE	0	unspecified
MMID_AIR	1	air
MMID_STD	2	span gas
MMID_ZERO	3	zero gas
MMID_PRES	4	pre-span
MMID_POSTS	5	post-span
MMID_PREZ	6	pre-zero
MMID_POSTZ	7	post-zero
MMID_PREB	8	pre-sample baseline
MMID_POSTB	9	post-sample baseline
MMID_BLINE	10	sample
MMID_SAMPLE	11	user-specified (16-bit)
<other>	-	

Measurement ID

Indicates measurement context/condition

mmeas syntax

```
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
```

queue API syntax

qpeek

```
 use    : qpeek I<id> [V] [out:<output_ctl>] [*] [ 0<=i<N ]  [ 1<=n<=N ]

brief   : read queue entries (non-destructive)

options :
  I : queue ID - queue index/handle 0:measurement queue 1: cycle queue
  V : verbose - show additional details where applicable
  * : all     - all queue entries
  i : index   - first entry
  n : entries - number of entries
  N : elements in queue (qcount)
```

qshow, qcount, qclear

```
use     : qshow|qcount|qclear [<id>]

brief   : qshow  : queue summary
          qcount : number of entries
          qclear : delete all entries

options :

  <id>  : queue ID - queue index/handle
          0:measurement queue 1: cycle queue
```

qpop

```
use     : qpop I<id> [VQ] [out:<output_ctl>] [*] [ 1<=n<=N ]

brief   : read and erase queue entries

options :

  I : queue ID - queue index/handle 0:measurement queue 1: cycle queue
  V : verbose - show additional details where applicable
  Q : quiet   - delete queue entries (no output)
  * : all     - all queue entries
  n : entries - first n queue entries
  N : elements in queue (qcount)
```

```
output_ctl - output control: out:[<dest>[,<fmt>]]

             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
```

Key terms

Licor record	A single Licor XML output
Observation	One or more Licor records (generally collected at the same time)
Measurement	Processed data and metadata for an Observation
Cycle	A set of related Measurements and metadata
Record	Data structure encasuplating a Measurement or Cycle
Log	File containing zero or more Records (typically binary format)
Queue	A generic stack mechanism for storing Records; a queue is manipulated using push/pop/peek semantics
Buffer (mbuf)	A container that holds a Record and acts as a circular queue for Measurements
Output device	A destination to which Record data may be sent : console, Queue, Log
mmeas	CWG command line API for measurement and output operations
qpush, qpop, qpeek	CWG command line API for queue operations