Los Gatos Research (LGR) Data Aquisition Program

Werner Eugster

Version from June 1, 2018

1 Introduction

This is a short user manual for the LGR data aquisition program `lgr` Version 1.08.

There are two programs in use: `lgrread` and `lgrshow`. The `lgrread` program performs data acquisition while `lgrshow` is only used as a user interface to monitor propper operation of `lgrread`.

2 Principle of Operation

The purpose of the `lgrread` program is to

1. receive data via a serial port
2. forward received data to
   (a) the sonic anemometer data aquisition program which saves the data on disk;
   (b) to a program or tool which allows the user to see what is going on and to detect problems and errors (e.g. `lgrshow`).

The forwarding is done via two System V Interprocess Communication (IPC) message queues.

3 Getting Started

To start the LGR program use the following command line

lgrread *[options]*

with the following options:

- -n no realtime scheduling
- -v to obtain verbose output (useful for tracking down bugs and errors in operation, but not for productive operation)
- -? to obtain the most current usage information about **licor**; the program prints out its known options and exits without doing any measurements.

The `-n` option was introduced in December 2003 when it turned out that data aquisition in raw mode (especially the sonic anemometer's data aquisition program) with realtime scheduling under RedHat Linux 9.0 on a 2.4 GHz Acer TravelMate laptop computer was so efficient that there was hardly any CPU time left for anything else. In such cases we can free the CPU by starting this program with the `-n` option.

It is easy to find out whether you need the `-n` option or not. If you do not use it and find that your computer has like frozen (does hardly react to keystrokes and such) as soon as you start this program, then you need the `-n` option be specified on the command line when starting this program.

It is not quite clear to me whether this is a bug in the 2.4 Linux kernel, or whether the realtime scheduling is indeed **much** more efficient under the newer kernels than what we experienced with

older ones. But there also people who believe that this has to do with the processor type and that e.g. an old 486 was much more efficient with serial port throughput than the new ones.
To monitor the correct operation of the program you have the following possibilities:

1. use `sonicshow` to control whether data from the LGR are received and saved to disk via the sonic data aquisition program, or use `lgrshow` to check the LGR alone (the latter works without a sonic anemometer);
2. if you started the sonic anemometer data aquisition program using its `-e` option, the LGR's status information will also appear in the output of `sonicshow` (see page 4 for details);
3. and you can check the log file of `lgrread` which you will find in the working directory where you started `lgrread` from (the file name ends on `.log`);
4. if you started the sonic anemometer data aquisition program using its `-e` option, then you will also find status information about the LGR's operation in the sonic anemometer's log file which is a file that is referenced to in the `sonicrc` resource file used by the sonic anemometer data aquisition program.

4 Message Queue Specifications

A key for each message queue is received from the `ftok()` function. The two arguments for `ftok()` are `QUEUEPATH` and `QUEUEID`. For `QUEUEPATH` we use `"/tmp"`, and for `QUEUEID` we use `'H'` for the message queue that connects with the sonic anemometer data aquisition program, and `'I'` for the message queue that connects with the user control tool. In both cases the rule is set that if the message queue is full, the program will continue to operate, but ignore new data, until the message queue is emptied from the connected process (if any). Both message queues are independent from each other. Thus, under normal operation, we would read the ID `'H'`, while we only read from `'I'` when we are actively working with the setup. Otherwise we do not care that the messages in queue ID `'I'` are discarded.

The message type generated has the following magic number: `0x1bac0004` (both message queues).

At compile time one can specify whether the raw data should be output to queue `'I'` or only one record per second. This can be decided by defining `CONTROLOUTPUT` in `licormsg.h` to be either `CONTROL_RAW` (full temporal resolution of data) or `CONTROL_1SEC` (one record per second only).

5 Data Formats

5.1 Data Format Expected from LGR FMA or FGGA

As of version 1.05 both comma delimited and space delimited data modes should work.

It autodetects FMA and FGGA data in a way that uses the FMA data format as the default and switches to FGGA format if more than 6 data variables are detected.

As of version 1.08 the newest LGR FGGA firmware is supported that delivers both the measured moist mole fractions for CH_4 and CO_2 and the dry mole fractions. In such cases we simply replaced the moist mole fractions by the dry mole fractions and do **not** record the moist mole fractions. Since these files have the same header as other FGGA files we should recall that only the Toolik Lake data used an older FGGA without dry mole fractions calculated, and all files before September 2016. The first files with dry mole fractions stored directly are those from the 2016 Rotsee campaign. For reference: the two mirrors of that FGGA had 10.906 μs and 7.514 μs ringdown times on 2016-10-07, and the calibration flag which is actually now a fit flag had the value 3.

Note that the log file shows an entry that helps you to unanimously identify the files (see Section 7).

5.2 How `lgrread` Processes Data And Sends It To IPC

The data format of the messages of type `0x1bac0005` in the message queue are simple ASCII text lines. Example:

```
LGR CH4 20999.7 CO2 4959070.1 H2O 92477998.0 CellP 13798.2 CellT 3458.5
Mirror_us_1 10890000.3 Mirror_us_2 7526450.2 Calib 3.0
```

The first number is the CH_4 concentration in 0.1 ppb. Note that for older instruments that do not have a conversion to dry mole fraction this concentration is the mole fraction in moist air. Since 2016 it is the dry mole fraction.

The second number is the CO_2 concentration in 0.1 ppb. Note that for older instruments that do not have a conversion to dry mole fraction this concentration is the mole fraction in moist air. Since 2016 it is the dry mole fraction.

The third number is H_2O concentration in 0.1 ppb

The text elements are invariant and are only for easier reference.

6 Lgrshow

> *NOTE: when you start `lgrshow` then the first lines you see are old lines that accumulated in the queue without anyone reading them. Only after those records are displayed you will be able to see new data. The easiest way of operation is to hit the <Enter> key on the keyboard and watch how the empty line moves up the screen one line every second. If this does not happen, then you either not running the `lgrread` data aquisition program, or there is some issue with the serial communication, maybe the output format of the data string is not correctly specified. The setting on the LGR instrument should be working with comma separated files.*

`lgrshow` displays one record per second no matter what the sampling rate is. The output looks like this:

```
FGGA CH4 2.0867 ppm  CO2 501.6020 ppm  H2O 9117.0704 ppm  P 137.98 Torr  T 34.24 C
Ringdown#1 10.895 us  Ringdown#2 7.530 us
```

It is essential that these values are correctly interpreted, otherwise they will **not** be stored correctly with the sonic data.

7 The Log File

`lgrread` has its own log file defined by `LOGFILENAME` in `lgr.h`. It notes the beginning of operation and all events that need to be protocolled. To still make it readable (especially when there are frequent communication errors), we only update the status of missed data at certain intervals as defined by `MISSED_LOG_INTERVAL` in `lgrlog.h` (currently set to 1 minute).

An example of a log entry is the following:

```
*****
Starting: lgrread Version 1.08 (07 October 2016)  Fri Oct  7 14:30:47 2016
Experimental access to LGR CH4 data Fri Oct  7 14:30:47 2016
Using device /dev/lgr for input and output Fri Oct  7 14:30:47 2016
 . now detected 11 LGR variables  Fri Oct  7 14:30:48 2016
 . Instrument type: FGGA (CH4/CO2/H2O) Fri Oct  7 14:30:48 2016
```

The log file is written in append mode and is closed immediately after an entry is written to it. That means that you should be able to move or rename the log file at any time to e.g. save it actively somewhere. If the original log file is gone, then the next time an entry is to be written a new log file with the **old** name is created and the logging is continued. That is, it is your duty to remember which log file is followed by what new file. The time stamps at the end of each entry facilitate this.

The autodetection of the instrument leaves a trace in the log file that helps to unanimously identify the file and instrument types. In the example above, there were 11 variables detected. This is 2 more than the older FGGA (9 variables) and means that in this case we saved the two extra variables, the dry mole fractions of CH_4 and CO_2 in place of the moist mole fractions (which were not saved).

7.1 Other Means of Controlling the Operation of the LGR

7.1.1 sonicshow

> *NOTE: when you start* `sonicshow` *then the first lines you see are old lines that accumulated in the queue without anyone reading them. Only after those records are displayed you will be able to see new data. The easiest way of operation is to hit the <Enter> key on the keyboard and watch how the empty line moves up the screen one line every second. If this does not happen, then you either not running the sonic data aquisition program, or it is not communicating with the sonic anemometer, or data do not arrive as expected!*

Because the data are passed to the sonic anemometer data aquisition program by an interprocess communication pipe, you can also monitor the LGR's actual status by using `sonicshow` (originally in the TOOLS directory of the sonic anemometer data aquisition program source code, now at the same level to avoid confusion).

`sonicshow` can be called by any user on the computer, it must not be the user who collects the data. This can be used to separate data aquisition (in superuser space!) from standard persons who just want to check the functioning of the equipment.

To exit `sonicshow` one just has to press <`Ctrl`> `C` on the keyboard (this stops `sonicshow`, but certainly not the data aquisition program).

An example of the screen output from `sonicshow` is the following:

```
0.00   -0.02    0.01  294.08   20.93   -0.50   -1.61 LGR   0
0.01   -0.03    0.01  294.09   20.94   -0.50   -1.61 LGR   0
0.01   -0.02    0.00  294.08   20.93   -0.50   -1.61 LGR  40
0.01   -0.02    0.00  294.08   20.93   -0.50   -1.61 LGR  40
0.01   -0.02    0.00  294.08   20.93   -0.50   -1.61 LGR 240
0.01   -0.02    0.00  294.08   20.93   -0.50   -1.61 LGR 200
0.02   -0.01    0.00  294.09   20.94   -0.50   -1.61 LGR   0
0.02   -0.01    0.00  294.08   20.93   -0.50   -1.61 LGR  40
0.02   -0.01    0.00  294.10   20.95   -0.50   -1.61 LGR   0
```

At the very end after the LGR characters there is the status of the LGR. It is not an instantaneous status information, but the summary since the last line was displayed. Each bit that was high at least once during that period will remain high until the status is displayed on the screen. The status information is in octal numbers and can be interpreted using the definitions in Section 5.

The example above shows the following: initially the LGR was working as expected (code 0), then the arrival of LGR data records was laging behind the arrival of sonic records (code 40), which is still OK. Recall that if we collect data asynchronously from two sources, the LGR and the sonic anemometer, at the same rate (e.g. 20 Hz) then it is no surprise that during some time the arriving data from the LGR arrives before the data from the sonic (code 0) or after it (code 40).

Any code above 40 is a problem, you need to find out with `irgashow` (see below). A code 2xx (e.g. 200 or 240) is the indication that no new data arrive from the LGR.

A Compilation Notes

In general this code should be compilable under a UNIX operating system. However, all the issues with byte swapping are and will not fully be tested on other systems than Linux running on an Intel-type Personal Computer (which uses low-endian byte order).

On Linux (Caldera OpenLinux 2.2) I used the `egcs` rpm package version 2.91.60-4 which was problematic. It will compile things, but if they are on a network drive or referenced to with a symbolic link, the compiled program won't work or it won't even link the various objects.

Then I update `egcs` to version 2.91.66-5 which works fine!

To compile the program, open `Makefile` in a text editor and make sure that only one line containing `CC =` is uncommented.

On newer Linux systems (e.g. RedHat 9.0) no such problems occurred.

I never tested this program under MacOS X but modified the code to be able to compile under MacOS X. Most important is that realtime scheduling is not available under MacOS X and had to be excluded. Thus it is not possible to test this part of the program under MacOS X, but my experience is also that newer faster computers do not even need this anymore. In any case, if you are ever going to try out this under MacOS X try it with very low overall system load since in such a case realtime scheduling is not needed at all.

For a running program under Linux (not a debug version) this command in `Makefile` should read something like

```
CC = gcc -D_SVID_SOURCE -O
DEBUGOPTS =
```

and for a debugging version it should read

```
CC = gcc -D_SVID_SOURCE
DEBUGOPTS = -g
```

There is also the possibility to define a precompiler variable `__TEST__` in order to use a standard input and output file for testing the program in place of the serial port.

A.1 Real-time operation

If `licor` is executed with super-user privileges, it will use memory locking and a higher process priority. Because we use the same module for the sonic anemometer, the IRGA and the FogMonitor, `realtime.h` might have to be adjusted. But also recall that there is now a `-n` switch to disable realtime scheduling. This still keeps memory locking and higher process priority active.

B Serial Port Specifications

All interactions with the serial port are in the module `serialport.o` with the source code files `serialport.c` and `serialport.h`. For easiest operation we do not specify the individual device address of the serial port, but a generic one named `/dev/lgr`. On the system, where this program is supposed to operate, we then set a symbolic link to the appropriate device and give it the appropriate permissions. We have to do this as root:

```
# cd /dev
# rm -f lgr
# ln -s ttyS0 lgr
# chmod a+rw lgr
```

`/dev/ttyS0` is the first serial port on old Linux systems. It seems that the `/dev/cua*` devices are obsoleted on newer versions and that one should now use `/dev/ttyS0` and subsequent ports. Subsequent serial ports have higher numbers (ttyS1, ttyS2, ...).

For more details consult Michael R. Sweet (1999) Serial Programming Guide for POSIX Operating Systems. 5th edition. http://www.easysw.com/∼mike/serial/index.html.

C Technical Notes

C.1 Data Format of N2O/CH4/H2O Analyser

The test with low-frequency records carried out on 2018-06-01 yielded the following format (after removing leading blanks and shrinking white space to single white space):

```
2018/06/01 11:07:06.325, 1.977320e+0, 0.000000e+0, 1.458955e+4, 0.000000e+0, 3.270479e-1, 0.000000e+0, 3.318900e-1, 0.000000e+0, 2.006596
2018/06/01 11:07:07.320, 1.976684e+0, 0.000000e+0, 1.457648e+4, 0.000000e+0, 3.269380e-1, 0.000000e+0, 3.317741e-1, 0.000000e+0, 2.005924
2018/06/01 11:07:08.315, 1.976518e+0, 0.000000e+0, 1.457251e+4, 0.000000e+0, 3.270397e-1, 0.000000e+0, 3.318760e-1, 0.000000e+0, 2.005747
2018/06/01 11:07:09.309, 1.976384e+0, 0.000000e+0, 1.461048e+4, 0.000000e+0, 3.278065e-1, 0.000000e+0, 3.326669e-1, 0.000000e+0, 2.005689
2018/06/01 11:07:10.303, 1.976426e+0, 0.000000e+0, 1.453642e+4, 0.000000e+0, 3.266287e-1, 0.000000e+0, 3.314468e-1, 0.000000e+0, 2.005580
2018/06/01 11:07:11.299, 1.976475e+0, 0.000000e+0, 1.460297e+4, 0.000000e+0, 3.274224e-1, 0.000000e+0, 3.322746e-1, 0.000000e+0, 2.005765
2018/06/01 11:07:12.293, 1.975491e+0, 0.000000e+0, 1.456953e+4, 0.000000e+0, 3.272265e-1, 0.000000e+0, 3.320646e-1, 0.000000e+0, 2.004699
2018/06/01 11:07:13.287, 1.975140e+0, 0.000000e+0, 1.457607e+4, 0.000000e+0, 3.271835e-1, 0.000000e+0, 3.320231e-1, 0.000000e+0, 2.004356
2018/06/01 11:07:14.282, 1.976913e+0, 0.000000e+0, 1.457116e+4, 0.000000e+0, 3.269112e-1, 0.000000e+0, 3.317451e-1, 0.000000e+0, 2.006145
```

One example record split into columns using the comma:

```
 1    Time              2018/06/01 11:07:06.325
 2    [CH4]_ppm         1.977320e+0          1.97732000    keep
 3    [CH4]_ppm_sd      0.000000e+0
 4    [H2O]_ppm         1.458955e+4      14589.55000000    keep
 5    [H2O]_ppm_sd      0.000000e+0
 6    [N2O]_ppm         3.270479e-1          0.32704790    keep
 7    [N2O]_ppm_sd      0.000000e+0
 8    [N2O]d_ppm        3.318900e-1          0.33189000    keep
 9    [N2O]d_ppm_sd     0.000000e+0
10    [CH4]d_ppm        2.006596e+0          2.00659600    keep
11    [CH4]d_ppm_sd     0.000000e+0
12    GasP_torr         4.501322e+1         45.01322000    keep
13    GasP_torr_sd      0.000000e+0
14    GasT_C            4.529731e+1         45.29731000    keep
15    GasT_C_sd         0.000000e+0
16    AmbT_C            4.719219e+1         47.19219000    keep
17    AmbT_C_sd         0.000000e+0
18    RD0_us            9.500000e-1          0.95000000    keep
19    RD0_us_sd         0.000000e+0
20    LTC0_v            -2.716303e-1        -0.27163030
21    LTC0_v_sd         0.000000e+0
22    AIN5              6.388015e-1          0.63880150
23    AIN5_sd           0.000000e+0
24    AIN6              1.027035e+0          1.02703500
25    AIN6_sd           0.000000e+0
26    DetOff            2.464076e+0          2.46407600
27    DetOff_sd         0.000000e+0
28    Fit_Flag          3                    3            keep
29    MIU_VALVE         -1                  -1
30    MIU_DESC          Disabled
```