Quantum Cascade Laser (QCL) Data Aquisition Program

Werner Eugster

Version from May 15, 2020

Contents

1 Introduction

This is a short user manual for the QCL data aquisition program `qcl` Version 2.05.

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

2 Principle of Operation

The purpose of the `qclread` program is to

1. receive data via a serial port **or** from an Aerodyne transient file (`-T` option)
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. `qclshow`)

(c) and to send data to another data acquisition if needed

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

NOTE: Currently each application is a specific configuration, and therefore the source code uses a set of magic numbers to determine the mode of operation. To change this mode, please change `QCL_INCOMING` in file `qcl.h` and recompile.

Version 2.04 is the first that allows to run more than one QC-TILDAS in combination with `sonicreadHS`. The version of `sonicreadHS` that can accept these data streams has yet to be written, but it most likely will be version 7.08 (check there; at some point we'll have to write a table to remember which version of which program is compatible with which other version of another program).

3 Getting Started

To start the QCL program use the following command line

qclread *[options]*

with the following options:

-?	to obtain the most current usage information about **qclread**; the program prints out its known options and exits without doing any measurements.
-n	no realtime scheduling
-v	to obtain verbose output (useful for tracking down bugs and errors in operation, but not for productive operation)
-s #	run as different system (started in version 2.04)
-T <transient_file.txt>	use a transient file instead of the serial port (started in version 2.01)
-c	channel output of non-EC data (for Davos ICOS only)
-F <format>	specify incoming data format use names in qcl.h, overrides QCL_INCOMING Example: qclread -F OPERATION_MODE_DAVOS_ICOS At Chamau: qclread -F OPERATION_MODE_CH4_N2O_H2O_T_P_2018 and then specify the order of variables with –ch4 etc. (see below)
--n2o	use variable at this position for N2O
--ch4	use variable at this position for CH4
--co2	use variable at this position for CO2
--h2o	use variable at this position for H2O
--no2	use variable at this position for NO2
--nh3	use variable at this position for NH3
--cellT	use variable at this position for cell temperature
--cellP	use variable at this position for cell pressure
--skip	ignore variable at this position

4 Command line options

4.1 -n : No realtime scheduling

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.

New computers (as of 2018) that have more than one CPU core do not have a problem. If you block one CPU for this task you most likely will still have 1 or 3 other cores available for other work, and hence realtime scheduling will be preferred. You can find out the number of CPU cores with the command

```
grep -i core /proc/cpuinfo
```

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 are 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 see whether a process is running with realtime privileges you can use the command

```
ps eaxlf
```

Among all other processes we find our data acquisition programs at sleep:

```
F   UID   PID  PPID PRI  NI    VSZ   RSS WCHAN  STAT TTY        TIME COMMAND
4     0  1124     1   3 -17   7552  3244 -      S<L  ?        3988:11 sonicreadHS -n -li -e1.6
4     0  1740     1   0 -20   4776  2492 -      S<L  ?         13:19 licor -n -M 7500
4     0 13135     1 -100  -   4372  2208 -      S<L  ?          0:08 qclread -F OPERATION_MODE_CH4_N2O_H2O_T_P_2018
```

The information in the STAT column is S<L; with man ps you can find out what this means:

```
S    interruptible sleep (waiting for an event to complete)
<    high-priority (not nice to other users)
L    has pages locked into memory (for real-time and custom IO)
```

Note, that our programs are mostly sleeping; 20 Hz is extremely slow for a computer running at 806,125,000 Hz . . . hence we use memory locking together with realtime scheduling to make sure the sleeping program is never paged out to harddisk (and then is not ready when data arrive).

4.2 -v : Verbose output

For troubleshooting and bugfixing we use the -v option that generates lots of outputs on the screen (verbose output) that we do not need under normal operation. Verbose output slows down a program considerable, thus under operational mode one should never use -v!

4.3 -c : Channeling output of non-EC data

The -c option was designated to the special case at ICOS Davos where we intend to use the same QC-TILDAS for eddy covariance flux measurements and alternatively for slow chamber concentration change measurements. In order to know whether we are in eddy flux mode or chamber mode, we read the valve position from qcl_status_word and potentially could guide the datastream to somewhere if the valve position is on QCL_VALVE_NONEC_VALID. This code was however never finished, and hence it does not have any effect as of 2018-12-07.

4.4 -F : Flexible-format incoming data

This option which takes one argument was introduced to override hardcoded settings of the incoming data format. The argument after -F is one of the following option listed in qcl.h:

```
#define  OPERATION_MODE_N2O                    1
#define  OPERATION_MODE_ISOTOPES               2
#define  OPERATION_MODE_ISOTOPES_PS            3 /* Patrick Sturm mode */
#define  OPERATION_MODE_CH4_N2O_NO2            4 /* August 2009 instrument */
#define  OPERATION_MODE_CH4_N2O_H2O_T_P        5 /* ETH REquip instrument */
#define  OPERATION_MODE_CH4_N2O_H2O_T_P_2018   6 /* Repaired Chamau QCL Dec 2018 */
#define  OPERATION_MODE_DAVOS_ICOS             7 /* Davos ICOS instrument */
```

The default is OPERATION_MODE_DAVOS_ICOS.

For the special case of the repared Chamau QC-TILDAS we had to solve the following problem: after revision the data stream was organized differently than before, and H2O had a better data resolution than before. To allow qclread to reorganize the data stream and bring the variables in the order that we had before repair, I specifically added support via

```
-F OPERATION_MODE_CH4_N2O_H2O_T_P_2018
```

and then each data column needs to be identified with the corresponding variable name using the variable name options described in Section 4.5.

4.5 Specifying variable names with long options

Long options in Unix are options that begin with two hyphens (--) instead of only one hyphen. Currently the following long options are foreseen:

--n2o	use variable at this position for N2O
--ch4	use variable at this position for CH4
--co2	use variable at this position for CO2
--h2o	use variable at this position for H2O
--no2	use variable at this position for NO2
--nh3	use variable at this position for NH3
--cellT	use variable at this position for cell temperature
--cellP	use variable at this position for cell pressure
--skip	ignore variable at this position

Thus to receive the data from the repaired Chamau QC-TILDAS and reformat them to the old CH4_N2O_H2O_T_P order, we have to specify the incoming order of variables like this:

```
--ch4 --h2o --n2o --skip --cellT --cellP
```

Note that currently each variable can only occur once, otherwise it is overwritten by the later occurrence. The exception is --skip which can occur more than once and simply tells qclread to ignore the column. Do not use this for empty columns! The empty columns (i.e. columns that have no data, no numbers, no text, no nothing between two commas) are ignored anyway, thus the specification is only addressing the data containing columns of the data stream.

The full command line options for the repaired Chamau instrument is thus

```
qclread -F OPERATION_MODE_CH4_N2O_H2O_T_P_2018 --ch4 --h2o --n2o --skip --cellT --cellP
```

and if sonicreadHS is running with the -e1.6 option, then we should record the specified variables at the correct position in the binary eddy raw data files.

Note that the -F OPERATION_MODE_CH4_N2O_H2O_T_P_2018 could also be last, but the order of the long options is **essential**!

5 Monitoring the operation of `qclread`

To monitor the correct operation of the program you have the following possibilities:

1. use `sonicshow` to control whether data from the QCL are received and saved to disk via the sonic data aquisition program, or use `qclshow` to check the QCL alone (the latter works without a sonic anemometer);
2. if you started the sonic anemometer data aquisition program using its `-e` option, the QCL's status information will also appear in the output of `sonicshow` (see page 9 for details);
3. and you can check the log file of `qclread` which you will find in the working directory where you started `qclread` 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 QCL'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.

Note: the transient file method is quite a hack suggested by Philip Meier, and most likely without Philip Meier it won't work elsewhere.

6 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 `'K'` or only one record per second. This can be decided by defining `CONTROLOUTPUT` in `qclmsg.h` to be either `CONTROL_RAW` (full temporal resolution of data) or `CONTROL_1SEC` (one record per second only).

7 Data Formats

7.1 How EMPA Sends Data From QCL

We started with a data format of the following structure

```
a3191492481.435,32.0803e+01,55.2803e+04,,,,,
```

where "a" starts a new line, followed by the Igor time variable, followed by a comma and then the concentration data. Concentrations are in ppb unless specified otherwise.

The columns are N2O (ppb), CO2 (ppb); during the Lägeren experiment (fall 2005) it was N2O (ppb), CO2 (ppb), H2O (ppb). It turned out that the CO2 concentration data were erroneous since a wrong absorption line had been chosen (the one for $^{13}CO_2$).

In August 2007 we resumed work on this program but with a next generation QCL system from Bela Tuszon (EMPA) for isotopic measurements. The data strings provided by Bela look similar, but the columns with concentration information are:

$^{12}C^{16}O^{18}O$ (ppb), $^{12}C^{16}O_2$ (ppb), $^{13}C^{16}O_2$ (ppb), Temperature (UNIT), Pressure (UNIT)
For simplicity we hard coded such settings. Check the definitions in `qcl.h`:

```
/*
 * IMPORTANT: To select how the incoming data are interpreted
 * define QCL_INCOMING with one of the definitions that have
 * magic numbers in the following list:
 */
#define OPERATION_MODE_N2O 1
#define OPERATION_MODE_ISOTOPES 2

#define QCL_INCOMING OPERATION_MODE_ISOTOPES
```

In the code we then use a `switch` statement to specify the corresponding action that is to be taken. We then need to recall that corresponding modification are also needed in the `sonicreadHS` code. There, we will use the extension switch `-e#.#` to specify different settings, since this specification will also be saved in the header of each file and will later help to clearly know what is inside a binary file (if we keep track of what we did; see the documentation to `sonicreadHS`, there is a table that should be updated for each new version and variant of such measurements that we add).

7.2 How Newer QC-TILDAS Send Data

The newer versions of Aerodyne QC-TILDAS can send cell temperature (in Kelvin) and cell pressure (in Torr) in the datastream. The concept however is that these two variables are appended to a fixed number of columns, whereof some may remain empty. Hence, the datastream as it arrives from the QC-TILDAS looks like this:

```
c3514784099.812,20.5090e+02,32.1969e+01,11.9216e+06,,,,,296.266,30.993
```

(example from Chamau, 2015-05-18). Version 2.00 of `qclread` decodes this data format. For practical purposes we kept the concept in version 2.xx as it was before, but the different QCL data block size allow to distinguish data files that include temperature and pressure from those that do not.

The essential point is that the receiving program (`sonicreadHS`) is the correct version and is started with the correct option to specify the data format. For the Aerodyne instrument at Chamau which sends CH_4, N_2O and H_2O concentrations, this is the `-e1.6` option (extension data version 1, variant 6) from `qclread` version 2.00 in combination with `sonicreadHS` version 7.06, and the files with only CH_4, N_2O and H_2O have a QCL data block size of 14 bytes, whereas those that include temperature and pressure have 22 bytes.

The combination of the extension version, extension variant and number of bytes in a data record must give a unique information that allows to autodetect the file format later for conversion and processing. A critical issue is that not all versions of `qclread` are compatible with all versions of `sonicreadHS` since we never were able to specify a universal concept that would achieve this (most likely such a concept is not even possible).

7.3 How `qclread` Processes Data And Sends It To IPC

The data format of the messages of type `0x1bac0004` in the message queue are simple ASCII text lines. Example:

```
QCL CH4 1943030.0 N2O 321276.0 H2O 12645800.0 TEMP 296266.0 PRESS 31561.0
```

Note that all values displayed here are multiplied by a certain factor and maybe an offset is involved. This allows to double-check what is sent to the receiving program (`sonicreadHS`) which simply converts the values to integers and saves them with the eddy covariance raw data. So if the first decimal is not showing a zero, then this means that with the current setting we're not saving the full resolution of the data (système finlandais how they saved Licor 7700 data in their intercomparison). If this ever happens, then the multipliers in `qcl.h` should be revised.

In this example, the first number is CH_4 concentration in 1/10 ppb.

The second number is the N_2O concentration in ppt.

The third number is the H_2O concentration in ppb.

The fourth number is cell temperature in 1/1000 K.

The fifth number is cell pressure in 1/1000 Torr (1 Torr = 1 mm Hg = 75 Pa).

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

The newest Davos ICOS system has two additional columns (34 data bytes) which are a status information and a vici valve position variable.

The Chamau data after instrument repair (files after 2018-12-07 15:30) have the H_2O multiplied by factor 1000 as compared to the version before repair.

In May 2020 it turned out that there was a wrong output in the H_2O column, and when the correct column was selected on 13 May 2020, it turned out that the values sent by the QCL WinTel software was already in ppb, hence multiplying by 1000 produced values in ppt that were overranging the range of a 4-byte integer variable. Thus, on 15 May 2020 we changed the code in `qcl.h` back to have a multiplier of 1.0 for H_2O again. This change became effective at 13:32:27 CET and affected the file with the name `2020051513.C00` at the Chamau site.

8 Qclshow

> *NOTE: when you start `qclshow` 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 IRGA data aquisition program, or it is not communicating with the IRGA, or data do not arrive as expected!*

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

```
QCL N2O 312.01001 CO2 422680.0
```

`qclshow` has options that may be used for nonstandard applications:

-r	display the raw output
-s #	display the data from system # instead of system 0
-y	display the data that is sent to the other data acquisition system (if any)
-?	to obtain the most current usage information about **qclshow**

In case we reorganize the data stream (as for the repaired Chamau data) the `-r` output is not really the raw output, but is the same as the data sent to `sonicreadHS`. To really see the raw incoming data stream one has to use the `-y` option which forwards the incoming data without changes to a potential other data acquisition system (e.g. for the Davos chamber system; but the code has not been finalized).

To see the unmodified incoming raw data stream from the QC-TILDAS use the command

```
qclshow -y
```

For example, the incoming data from the repaired Chamau QC-TILDAS look like this:

```
c3627047042.191,27.6253e+02,13.3115e+03,36.2674e+01,83.7652e+05,,,,,295.661,26.066
c3627047042.291,27.6848e+02,13.3115e+03,36.2449e+01,83.7572e+05,,,,,295.661,26.066
c3627047042.391,27.7258e+02,13.3115e+03,36.2827e+01,84.0440e+05,,,,,295.662,26.069
```

and the raw interpreted data look like this:

```
QCL CH4 2581930.0 N2O 356486.0 H2O 13305900.0 TEMP 295659.0 PRESS 26076.0
QCL CH4 2495780.0 N2O 354827.0 H2O 13305800.0 TEMP 295658.0 PRESS 26082.0
QCL CH4 2518230.0 N2O 355200.0 H2O 13306600.0 TEMP 295659.0 PRESS 26065.0
```

(it is just a clear and quiet evening, and the CH_4 and N_2O data have dramatically increased as the night sets).

9 The Log File

`qclread` has its own log file defined by `LOGFILENAME` in `qcl.h`. Since version 2.04 only part of the log filenames is specified with `LOGFILENAME`, and the number of the system and the `.log` extension will automatically be appended. 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 `qcllog.h` (currently set to 1 minute).

An example of a log entry is the following:

```
*****
Starting: ./qclread Version 1.00 (28 September 2005)  Wed Sep 28 14:33:26 2005
Experimental acces to EMPA QCL data Wed Sep 28 14:33:26 2005
```

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.

Newer versions of `qclread` add two more lines:

```
Using device /dev/qcl for input and output Mon May 18 09:19:04 2015
 . now detected  5 QCL variables  Mon May 18 09:19:04 2015
```

The second line indicates how many QCL variables actually were interpreted by the program. If you send 5 variables, but only 3 are detected, then this means that there was a parsing error most likely due to the representation of the empty data columns that the QC-TILDAS sends. Version 2.00 is the first where the data format including temperature and pressure was correctly recognised.

After the modifications made in version 2.05 we get these lines in the log file at Chamau:

```
*****
Starting: qclread Version 2.05 (07 December 2018)  Fri Dec  7 15:28:04 2018
Experimental access to AERODYNE QC-TILDAS data Fri Dec  7 15:28:04 2018
Using device /dev/qcl for input and output Fri Dec  7 15:28:04 2018
 . now detected  6 QCL variables  Fri Dec  7 15:28:04 2018
 . now detected  6 QCL variables : CH4 H2O N2O N2O TEMP PRESS  Fri Dec  7 15:28:04 2018
```

Thus, there is a bug in this entry to be solved in the next version . . . This relates to the variables detected in the incoming data stream but it does not yet treat the `--skip` specification, hence we have N2O twice. In reality the second occurrence of N2O should read “skip” or something similar. Would also be good to write to the log file which variables are forwarded in which order!

9.1 Other Means of Controlling the Operation of the QCL

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

At the very end after the QCL characters there is the status of the QCL. 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 7.

The example above shows the following: initially the QCL was working as expected (code 0), then the arrival of QCL 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 QCL 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 QCL 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 QCL.

In newer versions of `sonicshow` the format has slightly changed. This is an example from Chamau from 2015:

```
-3.25    0.47   -0.30  295.58   22.43    0.00    0.00 LI(1)   0 QCL001
-3.21    0.06   -0.12  295.74   22.59    0.00    0.00 LI(1)   0 QCL001
-2.23    0.69    0.27  295.69   22.54    0.00    0.00 LI(1)   0 QCL001
-2.41    0.91   -0.12  295.46   22.31    0.00    0.00 LI(1)   0 QCL001
-3.06    0.74    0.09  295.50   22.35    0.00    0.00 LI(1)   0 QCL001
-3.40    0.81    0.05  295.50   22.35    0.00    0.00 LI(1)   0 QCL001
```

There, the sonic anemometer does not have an inclinometer, hence these two columns are 0.00. Then, the output is ready for multiple IRGAs, and hence the number of the IRGA is given

in parentheses after LI. Here we only have 1 IRGA and hence the entry is `LI(1)` followed by the status code 0 (everything is fine). For the QCL we removed the space between `QCL` and the status code which is 001 in this example. This means: everything is fine, but the data rate of the QCL is not the same as that of the sonic anemometer, but still within the boundaries of what we expect (hence xx1). In reality, the QCL sends approximately 10 Hz data, but the sonic anemometer runs at 20 Hz temporal resolution.

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 `qclread` 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/irga`. 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 qcl
# ln -s ttyS0 qcl
# chmod a+rw qcl
```

If more than one QC-TILDAS is used, then the default system (or `-s 0` is explicitly specified) uses `/dev/qcl` to read data. System 1 will use `/dev/qcl1` etc.

`/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, ...). If USB to RS232 converters are used, they normally get the device names `/dev/ttyUSB0`, `/dev/ttyUSB1`, etc. on Linux systems.

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.