Sonic Anemometer Data Aquisition Program

Werner Eugster, ETH Zürich

Refers to Software Version 8.09
Document Date: May 3, 2019

Contents

1 Introduction

This is a short user manual for the sonic anemometer data aquisition program `sonicreadHS` which operates with the new Solent R3/HS sonic anemometer, the old R2 sonic anemometer, or the Campbell CSAT3 anemometer, version 8.05.

Starting with version 7.00 I omitted blocking read on the serial port under normal operation. We had a couple of issues when the sonic anemometer was correctly working at startup (where nonblocking read was used even in older versions), but later failed. This was no problem on standard computers or laptops, but on MOXA computers with the ARM RISC processor this tended to block the CPU and freeze everything. Without blocking read we have to be more careful with what additional load we allow on the computer, but experience shows that this software works quites smoothly even without blocking read.

2 Quickstart Notes

- For the InnoFarm project we need `sonicreadHS` version 8.04 in combination with `lgrread` version 2.00 **and** we must specify the `-e1.7` command line option to `sonicreadHS`.
- For ICOS Davos we need `sonicreadHS` version 8.09 in combination with `licor` version 3.02 **and** we must specify the `-li` command line option to `sonicreadHS`. Additionally, we should use an IRGA TCP/IP data stream buffer via the `-B` command line option, and to name the ASCII files for ICOS (30-minute data files according to the ICOS specifications) correctly, the `-F5` option must be used to set file number #5 for these additional files. The full command line with all options for ICOS is: `/usr/local/bin/sonicreadHS -n -li -e1.6 -F5 -B`

3 Principle of Operation

The purpose of `sonicreadHS` is to

- configure the Solent HS research sonic anemometer (including R3, R3A, R3-50, HS-50 probably also) or a Campbell CSAT3 anemometer (firmware 4.0 or higher), or an old Solent R2 or R2A
- receive data from the anemometer
- optionally receive data from the Fog Monitor via a message queue
- optionally receive data from the Licor 7500, Licor 7000, or Licor 7200 IRGA via a message queue
- optionally receive data from a second Licor via an additional message queue (since V. 7.03)
- optionally receive data from extra instruments (currently implemented for QCL data or LGR data) via a message queue
- check incoming records for errors (block numbering)
- save correctly received data on disk

- periodically open a new file to avoid too large files
- periodically send out data to a message queue for potential monitoring of the program's operation

A special feature was introduced in version 3.09 to support the cheaper Solent GILL R3-50 ultrasonic anemometer which does not have an inclinometer. We use the same program and settings as for the HS sonic anemometer, but if we detect that the records with the inclinometer data are missing, then we switch to the mode that is appropriate for the R3-50 sonic.

4 Getting Started

First of all, your Solent sonic anemometer must be put into binary communication mode. The manufacturer delivers the anemometers with the setting on ASCII mode, which we do not support. Run the DOS software (RCOM3) that came with the sonic anemometer one time and set the configuration to BINARY, and leave the programm. After this, your sonic should be ready for our data aquisition system.

To start the sonic anemometer data aquisition program use the following command line

sonicreadHS *[options]*

with the options specified in Table 1.

Note that `sonicreadHS` reads a resource file **sonicrc** in the current working directory to find the settings it should use. This resource file is re-read every time the internal alarm clock expires to open a new file. To force a re-read of **sonicrc** one needs to send a `SIGALRM` signal to the running process. The easiest way to do so is via the use of a simple script, **newfile** which should be installed on the computer. `newfile` finds the process ID of the running program and asks you whether it should force this process to re-read its resource file. You need to reply with **yes** to see any action. Any abreviations or other answers are ignored and no changes to the running status of `sonicreadHS` will be effectuated.

4.1 POWERCONTROL Option

There are rare cases where a Gill/Solent sonic of the newer model may be stuck in interactive mode and the baud rate appears to be undeterminable. This condition can only be solved with a hard power reset of the instrument. This can also be done by compiling the code with `POWERCONTROL` set to `ON` in the Makefile. If this is active, an additional serial port defined via `SONIC_CONTROLPORT` in `scntl.h` can be used to hook up a relay circuit that does a 10-second hard power reset of the sonic anemometer during startup. By default we use the symbolic device `/dev/sonic-control`.

The support of CSAT3 sonics with this feature is not yet done, but should be added in a future version.

5 Running the Program More Than Once

In Version 5.01 we introduced a new concept that became necessary to run the same data acquisition programs more than once on one single computer, e.g. for intercomparison of flux instruments. This requires that different serial ports are used by the instruments, and the data are exchanged via specific interprocess communication queues that connect each instrument with its corresponding sonic anemometer data acquisition. This is done via the new `-s` switch, followed by a number from 0 to 9 (in principle, other single characters may also work).

For backward compatibility, `-s0` is the default and uses the standard naming of serial ports and IPC queues as it was the case with earlier versions. That means, that the sonic anemometer is read from `/dev/sonic`, the IRGA from `/dev/irga` and so on.

For other numbers, these device names are expanded by the same character (number). E.g. when we start our data acquisition as

```
sonicreadHS -s1
```

then this will read from `/dev/sonic1`, and screen display must be made with `sonicshow -s1` (otherwise it will read from the default queue which corresponds to `-s0`).

The device names are normally links to the correct devices on the system, e.g. made as root via

```
ln -s /dev/ttyS17 /dev/sonic
ln -s /dev/ttyS18 /dev/sonic-control
```

Table 1: Command line options of `sonicreadHS`.

Option	Explanation
-A	special features for ICOS analog tests
-B	install a 20 record stream buffer for the IRGA
-c	save SONIC raw data in separate file in working directory
-d	only with CSAT3: switch ON delimiter mode (delimiter ON)
-e#.#	receive data from an extra instrument using an extension versioning system; version is one digit from 0 to 7 before the decimal point, variant is a one-character hex code from 0 to f. See Table 3 for the correct numbering
-F #	save ICOS data in files using ¡number¿ in the FFN name component
-fm	receive Fog Monitor data blocks from the Sys V IPC message queue and append them to the anemometer data, and save them on disk. If this option is specified, the file type will be **'F'** (see Table 2).
-f	is synonymous to `-fm`
-I#	allows to also receive IRGA data from a **secondary** IRGA that runs with the specified system number; these data will added to the data stream immediately after the ones of the default IRGA
-i	is synonymous to `-li`
-li	receive IRGA data blocks from the Sys V IPC message queue and append them to the anemometer data, and save them on disk. If this option is specified, the file type will be **'I'** (see Table 2).
-l	is synonymous to `-li`
-M XXXX	(with XXXX = CSAT3 or METEKu3) use the CSAT3 communication protocol for Campbell CSAT3 sonic anemometers running firmware 4.0 or higher, or the METEKu3 protocol for a METEK μ3 sonic anemometer
-m	is synonymous to `-fm`
-n	no realtime scheduling; this switches off the realtime scheduling priority which could block a newer computer/linux installation for all other processes (this is the switch I have to use with kernel 2.4.20-8 from RedHat Linux 9.0 on an Acer TravelMate 242LC with 2.4 GHz processor speed)
-o	switch off the delimited mode of the CSAT3 sonic; only has an effect in combination with -M CSAT3. Currently the software has not yet succeeded to switch the sonic to delimited mode and use it correctly, **thus you need to specify -o when working with a CSAT3!**
-R2	use the Solent R2 (fastcom) communication protocol; note that this only works if the internal jumpers on the motherboard of these old sonics is set to the position to deliver single records (default is to send blocks at 20 records which is not supported by this data acquisition program here)
-R3	use the default Solent HS or R3 protocol (works with all newer solent sonics that come with an RCOM3 software diskette for DOS operating systems); this switch is ignored since this is the default
-r	read-only mode. This allows to bypass the configuration step, so that `sonicreadHS` can work with a sonic that cannot be configured. It however requires that you tune `sonicreadHS` in a way that its settings match the true settings that the sonic actually is running with. Check your `sonicrc` file.
-s#	run as system number 0 to 9; this allows to run several instances of the same program on the same computer for more than one sonic anemometers and additional sensors. -s0 is the default and is ignored (default behavior conforms to what we used in older versions)
-u	this is an experimental and unsafe option which tries to unlock the sonic anemometer in case there is no continuous data stream at the serial port. But because there are many possibilities why there are no incoming data, it often does not save you the trip to the power switch of the sonic anemometer to switch it off and on again physically. However, with the `POWERCONTROL` option active, you can use a relay to do a physical power reset on the sonic anemometer.
-v	produce verbose output (good only if there are problems to be tracked down).
-x	configure sonic for the first time (from factory settings), normally used when the Gill sonic is set to ASCII data output mode instead of BINARY

Note that if `/dev/sonic-control` is not used, then it should be linked to `/dev/null`. This is a serial port that allows to do a hard power reset on the new Solent sonics which have the conceptual error that they can get stuck in interactive mode at unknown baud rate which never times out. This feature is only available with special electronics (a relay switched via one of the handshaking lines of that second serial port) and is most likely not used by others.

6 File Header

The file header (Section 6.1) corresponds to the definition given in the user manual of the Solent HS research ultrasonic anemometer (Doc. No. 1199-PS-0003-Iss 4) unless we use a CSAT3 instrument with the -M CSAT3 option (Section 6.3).

6.1 WECOM3 Header Format

The **only** but important difference is the labeling of the file type.

The header contains a member `file_type` which can have the values given in Table 2.

Table 2: File types of Solent R3/HS sonic anemometer data.

Value	Meaning	Defined by
0	ASCII tabular	Solent user manual
1	Binary tabular	Solent user manual
2	ASCII micro-met	Solent user manual
3	Binary micro-met	Solent user manual
A	Werner Eugster's binary format without FogMonitor data	here
F	Werner Eugster's binary format with FogMonitor data	here
G	Werner Eugster's binary format with FogMonitor and IRGA data	here
I	Werner Eugster's binary format with IRGA data	here
K	Werner Eugster's binary format with IRGA data from 2 systems (since V. 7.03)	here
other	Werner Eugster's binary format (see below)	here

To allow for more special instruments for flux measurements we introduced a more general "extension" that uses bits 5 and 4 of the file type byte. Thus, the concept is that we by now have the two instruments FogMonitor and IRGA that we want to use as standard instruments in any combination with the sonic, and if we add more instruments, we'll do this via the extension option. Because we only have 3 bits left in this header byte, and with `sonicreadHS` version 8.0 we want to use bit 7 to mark the new version 2 sonic data format, we can use an **extension version** in the range 0 to 3 (bits 5 and 4), where 0 is defined as the setup without any extensions, and 1–3 can be used in fact. This is not perfectly flexible, and thus we use a **variant** in addition to the extension. The variant is not saved in the header, but in the individual records. This will allow to use more than one extra instruments at the same time, the only issue is that we need to keep track of what the numbers mean such that we can learn our data processing software what to do with the data.

So far we used extension version 1, variants 0–2 for EMPA data from the quantum cascade laser system, and extension version 1, variants 3 and 4 for the two Los Gatos Research analyzers. If we're going to use this concept with Joseph McFadden's CO analyzer we'll call this extension version 2, variant freely chosable. To keep track on what we did see Table 3.

6.1.1 Freeform extension data option

In `sonicreadHS` version 8.05 support for option `-e1.f` was introduced, which is the "freeform" option that should simplify test setups for which we leave it to the user to keep track of what's found in the extension data. When specifying the `-e1.f` command line option the string messages that arrive via the LGR message queue are not interpreted, but simply stored as a character string in the raw data files. The concept is still the same that 2 bytes will be saved under all circumstances, the first specifying the length of the data record (number of bytes to be read when reading a binary data file), and the second byte being a status flag information. The number of bytes are no longer a fixed number or 2, since the number of bytes will be determined from the length of the message received via the message queue. First application was with the LUMEX experiment where we tried to measure Hg^0 fluxes for the very first time

Table 3: Extension versions and variants.

	Variant	Content	Columns
Version 1			*reserved for W. Eugster*
	0	QCL EMPA	N2O, CO2, H2O
	1	QCL EMPA	12C16O18O, 12C16O2, 13C16O2
	2	erroneous IRGA data from first test	could be re-used
		QCL Sturm&Knohl	12C16O18O, 12C16O2, 13C16O2, CO2 calibrated, δ^{13}C, δ^{18}O
	3	LGR FMA CH4 data and house-keeping variables	CH4, Cell-P, Cell-T, Mirror Ringdown time, Calibration flag
	4	LGR FGGA CH4, CO2, H2O data and house-keeping variables (since version 5.12)	CH4, CO2, H2O, Cell-P, Cell-T, Mirror 1 Ringdown time, Mirror 2 Ringdown time, Calibration flag
	5	QCL EMPA	CH4, N2O, NO2, H2O
		to reduce the number of variants, the following formats were hardcoded in 1.6 they can be distinguished based on the number of bytes in the record	
	6	QC-TILDAS ETHZ	CH4, N2O, H2O until V. 7.05
	6	QC-TILDAS ETHZ	CH4, N2O, H2O, TEMP, PRESS since V. 7.06 (2015-05-18)
	6	QC-TILDAS ETHZ ICOS Davos	CH4, N2O, H2O, CO2, TEMP, PRESS, StatusW, VICI_valves since V. 7.07 (2015-11-13)
	7	LGR CH4/N2O/H2O	see Table 14; first used with InnoFarm in 2018 (sonicreadHS version 8.04, lgrread version 2.00)
	8	unused	
	9	unused	
	a–e	unused	
	f	**freeform** – generic solution that stores the text data string from the message queue "as is"	depends on what the supply program (`qclread`, `lgrread`, ...) delivers
Version 2			*reserved for W. Eugster and collaborators*
	0–1	unused	
	2	QC-TILDAS from **two** data streams (since V. 7.09 for Lutz Merbold's Mazingira project)	Format according to `qclread` V 2.04 or higher. Uses component-specific conversion, see Table 11.
	3–9	unused	
	a–f	unused	
Version 3		unused	

and simply sent the date and timestamp from the embedded windows system of the LUMEX analyzer to `sonicreadHS`. Since the extension data will be recorded in ASCII format "as is" they should be readable and indicate what file this is, in case information was lost what variant 1.f files contain.

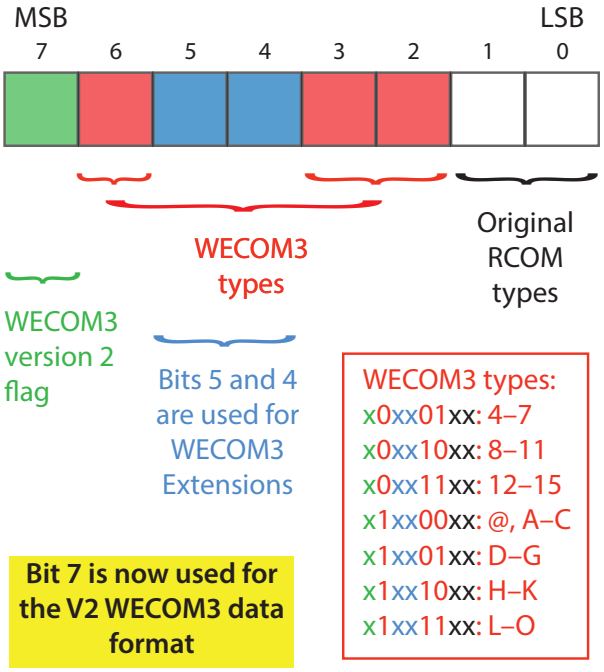

Figure 1: Bits of the `file_type` byte. Bit 7 was redefined with version 8.00 of `sonicreadHS`, but was never used before this redefinition.

6.2 Important update with `sonicreadHS` version 8.00 and higher

The bits in the `file_type` byte were reserved as in shown in Figure 1. Since we have only used extension versions 1 and 2 so far, I made the decision to redefine bit 7 to mark version 2 WECOM3 data. Thus, we reduce the possible extension versions to 1–3 (we still have many variants that we can use). Since this redefinition of bit 7 does not affect any data files that we have collected so far, I modified the text above on 2017-07-03 to reflect the new data format definition.

6.3 WESAT3 Header Format

For support of CSAT3 sonics we also changed the header format, but the concepts for the WESAT3 files is identical to the WECOM3 files apart from the difference in header and difference in bytes holding the sonic anemometer data. This header is 38 bytes. For details see the documentation of the `eth-flux` data processing software.

7 The Log File

The log filename is specified in `sonicrc`. If there is no `sonicrc` file, the default will be something like `sonic.log`. An example is given below:

```
***** STARTING sonicreadHS version 2.32 from 25 February 2000   Fri Feb 25 12:07
:50 2000
Message queue 128 installed for reading FogMonitor data.        Fri Feb 25 12:07
:50 2000
unable to find sonic at baud rate 15    Fri Feb 25 12:08:03 2000
found sonic at baud rate 17     Fri Feb 25 12:08:04 2000
found sonic at baud rate 17     Fri Feb 25 12:08:05 2000
found sonic at baud rate 17     Fri Feb 25 12:08:07 2000
```

```
found sonic at baud rate 17     Fri Feb 25 12:08:27 2000
Sonic HS serial number is 000027        Fri Feb 25 12:08:28 2000
Sonic HS sampling rate (average 10) is 10.0 Hz  Fri Feb 25 12:08:29 2000
Sonic internal software version 2.01 2076_201 R3>       Fri Feb 25 12:08:29 2000
found sonic at baud rate 17     Fri Feb 25 12:08:30 2000
      alarm signal received 0 0 0 0     Fri Feb 25 12:08:30 2000
User settings: data path "./" file code "b"     Fri Feb 25 12:08:30 2000
mode 1 analogs 0 baud 17 changefile 6 hours     Fri Feb 25 12:08:30 2000
opening data file ./2000022512.b09      Fri Feb 25 12:08:30 2000
next file scheduled in 24678 seconds    Fri Feb 25 12:08:30 2000
new FM status:  202 118 Fri Feb 25 12:08:31 2000
new FM status:   42 118 Fri Feb 25 12:09:15 2000
new FM status:   42 118 Fri Feb 25 12:09:59 2000
new FM status:  242   2 Fri Feb 25 12:10:00 2000
```

Important are entries where there were record numbering problems:

```
record numbering problem 9 -> 186 (10, 1)       Fri Feb 25 15:47:26 2000
      alarm signal received 0 0 0 0     Fri Feb 25 15:47:26 2000
```

We'll have to keep an eye on this. It seems that there are some interferences with operating two high-frequency instruments at the same time. It could be that we will have to look for the Real-Time Linux extensions some day.

7.1 Interpretation of the Log File

```
***** STARTING /var/hda/progs/sonicreadHS version 7.06 from 18 May 2015 Mon May 18 10:53:17 2015
```

Whenever the program is started, it leaves such an entry in the log file. The date and time given on the far right side is always the timestamp when this happened. The information `version 7.06 from 18 May 2015` shows which version of the program is run (the number 7.06 is hard-coded in the source code), and when this version was compiled. Hence there may be different compilation dates for the same program version; this may be necessary if the same program is run on different computer platforms, which normally means that one recompiles the source code for that platform. For example, a MOXA embedded computer runs an ARM X-Scale processor and thus programs must be compiled for ARM X-Scale processors. At other sites Intel processors may be in use, and hence the program version is compiled for Intel processors, either for 32-bit operating systems or 64-bit operating systems.

```
Data files will use extension version.variant 1.6 Mon May 18 11:01:20 2015
```

If data from a QC-TILDAS or from a LGR FMA/FGGA or similar are added to the data from the sonic anemometer, then such an entry is made which specifies the version/variant number that you specified at startup with the `-e` switch. This information is also found in the header of each binary data file, so this is simply redundant information for your archive.

```
Message queue 0 installed for reading primary IRGA data. Mon May 18 11:01:20 2015
Message queue 98307 installed for reading QCL data. Mon May 18 11:01:20 2015
```

Each additional sensor sends its data via a Unix System V message queue. Here you see the queue numbers that are used to receive the data at full resolution, for the IRGA (Licor 7500, Licor 7000, Licor 7200 etc.) and for the QC-TILDAS (QCL).

```
Sonic power control signal is DTR on device /dev/sonic-control Mon May 18 11:01:30 2015
```

There is the option to have a power-reset relay installed to repower the sonic anemometer at startup. If this option is used and configured, this entry shows you which serial port is used and which signal (DTR) to activate the relay to interrupt power.

```
unable to find sonic at baud rate 16 Mon May 18 11:01:34 2015
found sonic at baud rate 17 Mon May 18 11:01:35 2015
```

Each sonic anemometer model has one or more baud rates at which they may communicate. At startup the programm cycles through the baudrates that were specified in the source code to find the sonic. The baud rate numbers are given in magic numbers as they are assigned in e.g. `/usr/include/sys/termios.h`. Under Linux 15 means 9600 baud, but on MacOS 9600 is shown here. 16 means 19200 under Linux, and 17 is 38400 baud, but be aware that these magic numbers really depend on the system. The important point is that at least one entry `found sonic` must be seen to have confidence that the sonic data stream was correctly arriving at the data acquisition serial port.

```
Sonic R3-50 serial number is 000348 Mon May 18 11:03:27 2015
Sonic HS/R3/R3-50 sampling rate (average 5) is 20.0 Hz Mon May 18 11:03:28 2015
Sonic internal software version 3.01 2076_301 Feb Mon May 18 11:03:29 2015
```

If communication with the sonic anemometer can be established, the serial number (for Gill sonics only) is stored in the header of the data files and in the log file. Additional information on how the data stream is interpreted follows in the log file. The sonic anemometer is configured according to the settings in `sonicrc`, and the sampling rate is explicitly documented in the log file (20.0 Hz in this example).

```
alarm signal received 0 0 0 0 Mon May 18 11:03:33 2015
```

Once the data acquisition starts, all modifications are made via alarm signals. Hence, and alarm handler is installed, and the four figures (0 0 0 0) show the status of four flags that help debug issues that may arise.

```
User settings: data path "./" file code "b" Mon May 18 11:03:33 2015
mode 1 analogs 0 baud 17 changefile 6 hours Mon May 18 11:03:33 2015
opening data file ./2015051811.b05 Mon May 18 11:03:33 2015
next file scheduled in 10555 seconds Mon May 18 11:03:33 2015
```

If everything is ready for data to be stored on disk, a data file is opened and the data start streaming to that file at zero seconds of the time and date specified in the file name and in the header of that file. So, if the file name sais `2015051811.b05`, then this means: first data record (in binary format) in this file has the time stamp 2015-05-18 11:05:00.00 local time. In Switzerland local time is set to CET (Central European Winter Time).

```
new IRGA status:    0  16 Mon May 18 11:03:33 2015
new IRGA status:  200   2 Mon May 18 13:15:08 2015
new IRGA status:    0  16 Mon May 18 13:15:09 2015
new QCL status: 011 ( 22 bytes) Mon May 18 11:03:33 2015
new QCL status: 001 ( 22 bytes) Mon May 18 11:03:34 2015
```

During operation the changes in the status of the status byte of the additional instruments (IRGA, QCL, ...) is recorded in the log file. The status is an **or** combination of the status bits received turing an entire second. The exmple above shows a case where no data were received from the IRGA on May 18 at 13:15:08 (no data means: no data during a longer period than what we hard coded for the typical behaviour of that instrument), but the second after the condition went back to OK.

```
No inclinometer present (this might be a GILL R50 anemometer) Mon May 18 11:03:34 2015
```

This entry is seen if no inclinometer data seem to arrive. For the sake of a consistent data format we hence store zeros in place of the inclinometer data in the raw data file.

7.2 Other Means of Controlling the Operation of the FogMonitor

Because the data are passed from the Fog Monitor to the sonic anemometer data aquisition program by an interprocess communication pipe, you can also monitor the FogMonitor's actual status by using `sonicshow` (in the TOOLS directory of the sonic anemometer data aquisition program source code).

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 FM   0
0.01   -0.03    0.01  294.09   20.94   -0.50   -1.61 FM   0
0.01   -0.02    0.00  294.08   20.93   -0.50   -1.61 FM  40
0.01   -0.02    0.00  294.08   20.93   -0.50   -1.61 FM  40
0.01   -0.02    0.00  294.08   20.93   -0.50   -1.61 FM 240
0.01   -0.02    0.00  294.08   20.93   -0.50   -1.61 FM 201
0.02   -0.01    0.00  294.09   20.94   -0.50   -1.61 FM   0
0.02   -0.01    0.00  294.08   20.93   -0.50   -1.61 FM  40
0.02   -0.01    0.00  294.10   20.95   -0.50   -1.61 FM   0
```

If you started `sonicreadHS` without the `-fm` option, you will not find the letters `FM` and the status information at the end of the lines, but the rest will be exactly the same.

7.3 Other Means of Controlling the Operation of the IRGA

The same applies to IRGA data collected with the `-li` switch. Just the letters `LI` followed by the status information is shown. Most important to know is that status 0 is always OK and 40 is OK depending on configuration, whereas all other numbers indicate a problem (e.g. 200 indicates that no data is coming in from the IRGA). Status 40 means that the records from the IRGA are not new. This can e.g. happen when Sonic and IRGA both run at 20 Hz, but the sonic is slightly ahead of the IRGA. This means that the current sonic record is merged with the previous IRGA record. Since we will determine the time lag later, this will just affect this time lag by one in this special case. Another case that is OK is when we have the sonic running at 20 Hz but the IRGA is only running at 10 Hertz. Since we only display 1 record per second in `sonicshow` we decided to show the worst status of any records that was received within the last second. Thus, although only every second record from the sonic should not have a matching IRGA record, we'll get status 40 although this is perfectly OK in this case.

8 The Resource File

The resource file is a simple ASCII text file. Each line begins with a token which can be interpreted by `sonicreadHS`. Valid tokens are: COMMENT, DATAPATH, ANALOGS, MODE, BAUDRATE, CHANGEFILE, FILECODE, LOGFILE, and AVERAGE. An example is given below.

```
COMMENT     This is a sample sonicrc file by Werner Eugster
DATAPATH    ./
COMMENT     the old sonic anemometer has 5, the new HS sonic 6 analog channels
ANALOGS     0
COMMENT     MODE is ignored; leave it with a 1
MODE        1
COMMENT     the old sonic anemometer works with 4800, 9600 or 19200
COMMENT     the new HS sonic anemometer works with 9600, 19200 or 38400
COMMENT     the CSAT3 sonic anemometer only works with 9600 (19200 does not work)
BAUDRATE    9600
COMMENT     CHANGEFILE is an integer; after ... hours a new file will be opened
CHANGEFILE  6
COMMENT     FILECODE is one character that will appear in the filename ext.
FILECODE    b
LOGFILE     myeddyfluxsystem.log
COMMENT     below are additional settings for the new Solent HS sonic
COMMENT     with the 50-Hz (cheaper) sonics it is as follows:
COMMENT     AVERAGE 1 is 50 Hz, AVERAGE 3 is 16.7 Hz, AVERAGE 5 is 10 Hz
AVERAGE     5
COMMENT     below are additional settings for the CSAT3 sonic
COMMENT     sampling rate of the CSAT3 is specified via a execution parameter
COMMENT     which is one character from 1,...,9,a...h
COMMENT     60 Hz averaged to 10 Hz is g
COMMENT     60 Hz averaged to 20 Hz is h
COMMENT     point sampling (no averaging): 10 Hz = 9, 12 Hz = a, 15 Hz = b, 20 Hz = c
```

```
EXEPARA     h
COMMENT     define where IRGA data should be tanken from for averaging
COMMENT     valid options are: ANALOG, DIGITAL, BOTH or NONE
IRGA        BOTH
COMMENT     IRGA 2nd order conversion from millivolts to concentration
COMMENT     introduced in version 3.01; conc = a0 + a1*x + a2*x^2
COMMENT     format: three floating point numbers for a0, a1, and a2
IRGA_CO2    0.0 1.0 0.0
IRGA_H2O    0.0 1.0 0.0
```

COMMENT lines are ignored. Also the MODE setting is ignored (we always use calibrated mode). Important settings are the baudrate settings (BAUDRATE), and the sampling rate setting (AVERAGE).

The CHANGEFILE setting takes an integer number of hours, afte which a new file will be opened (to keep file size reasonable). If you specify 6 hours, this means that at 00:00 UTC, 06:00 UTC etc. you will get a new file. Thus, in local time new regular files will start 01:00 MEZ or 02:00 MESZ etc.

The FILECODE setting takes a single character (e.g. the letter **b**) which is incorporated in the filename creation. File names contain date information and the time when data collection started. An example with FILECODE **b** looks like this:

`2000022517.b09`

The format is YYYYMMDDhh.#mm, where YYYY is the 4-digit year, MM is the 2-digit month, DD is the 2-digit day of the month, hh is the 24-base hour of local (system) time, # is the FILECODE, and mm is the minutes in the our. The program always waits until the beginning of that minute with saving data. It creates the 29-byte header in due time, then sends old data to the old file until 0 seconds of minute mm start, where the old file is closed and the data are appended to the new file.

The ANALOGS setting indicates how many of the internal analog inputs of the sonic anemometers are used. `sonicreadHS` only supports a range of channels that start with the first channel (thus no possibility to individually switch on or off a channel). A value of 0 indicates that no analog input channels should be used. A value of 3 indicates that channels 1, 2, and 3 are switched on and recorded.

The BAUDRATE setting is typically 38400 (38400 baud) for the new HS sonic. Use a value that is valid for the sonic anemometer!

The AVERAGE setting indicates how many internal records are averaged to produce one output record by the sonic anemometer. This influences the sampling output rate of the sonic anemometer. For example, to yield 20 Hz with a Solent HS sonic (internal sampling at 100 Hz) this value is set to 5.

The IRGA_CO2 and IRGA_H2O settings are new in version 3.01 and indicate how internal values are converted to concentration readings for the online output of average values. This is only meant for control purposes. To get correct concentration and flux values a post-processing of the raw data is still necessary. Three floating point numbers separated by one or more white space characters must be given for the coefficients a_0, a_1, and a_2 of the second order transfer polynome $c = a_0 + a_1 \cdot x + a_2 \cdot x^2$, where x is the millivolt reading of the respective channel and c is the concentration value. Default values are: $a_0 = 0.0$, $a_1 = 1.0$, and $a_2 = 0.0$.

A Binary WECOM3 Raw Data Format

The binary raw data formats **0**, **1**, **2**, and **3** are described in the Solent Ultrasonic Anemometer's User Manual (RCOM3 format).

The additional formats introduced by Werner Eugster (currently **A**, **F**, **G**, **I**, **K**) use a similar format but with a few changes and extensions described here.

A.1 Header

Each binary file contains a header of 29 bytes as described in the Solent Ultrasonic Anemometer's User Manual. The bytes have the following meaning (copied from the `convertall` source code):

```
typedef struct /* the new HS header size is 29 bytes */
{
unsigned char file_type;
unsigned char file_version;
unsigned char type;
unsigned char serial_number[4];
unsigned char average;
unsigned char wind_report_mode;
unsigned char string_format;
unsigned char ascii_terminator;
unsigned char echo;
unsigned char message_mode;
unsigned char conf_tone_setting;
unsigned char axis_alignment_setting;
unsigned char speed_of_sound_report_mode;
unsigned char abs_temp_report_mode;
unsigned char an_input_1;
unsigned char an_input_2;
unsigned char an_input_3;
unsigned char an_input_4;
unsigned char an_input_5;
unsigned char an_input_6;
unsigned char an_output_scale;
unsigned char an_output_wrap;
unsigned char file_create_time[4];
} HEADERTYPEHS;
```

As in RCOM3 the data itself are saved in big endian format, that is, the first byte of a multi-byte integer number is always the most significant. However, be aware that there are differences to this in the header, which are inconsistent, but have been set into the world by Gill/Solent. Changes compared to the original header of the RCOM3 files:

- **type**: in addition to 0–3 one finds the new file type code here. My type **A** is saved as decimal character 65, type **F** is decimal character 70, type **G** is decimal character 71, type **I** is decimal character 73 (corresponding to the position of the capital letters in the ASCII character set).
- **file_create_time**: in my variant of this binary format I did use the time stamp in local time, that is, this 4-byte long integer gives the seconds since 00:00:00 local time, 1 January 1970. To my understanding, the RCOM3 format is using time in UTC, thus depends on the time zone you have selected on your computer while doing the data aquisition. I have also used the little endian encoding also for the time stamp as in RCOM3, but be aware that all other data are saved in big endian format!
- **file_version**: When we record data from an old Solent R2 or R2A sonic (`-R2` command line option of `sonicreadHS`) then we find the values 99 or 98 in the `file_version` field, and `serial_number` is set to the value 0x09090909. The file_version 99 has two issues that need to be known: (a) after the base data (wind, speed of sound, analog channels) there might be an extra byte that should not be there, followed by 4 bytes with 0x00 that correspond to the inclinometer variable

that is however not available from R2/R2A sonics. In `sonicreadHS` version 5.07 we modified this and eliminated the erroneous extra byte from R2 sonics, and omitted the 4 empty bytes of the inclinometer variables.

- **serial_number**: the magic decimal value 151587081 (or 0x09090909) indicates that we were using an old R2 or R2A sonic. These do not provide their internal serial number in the communication protocol, therefore we used such a magic number to label the files in addition to the `file_version` number.

If unsure about the encoding (litte endian or big endian): bytes number 4–7 (starting counting with byte number 1) contain the serial number which typically is a small positive number with a lot of leading zeros. At LäŁgeren, for example, I used the instrument with serial number 27.

A.2 Data

After the header bytes the data records follow. The concept is as follows: the standard data arriving from the sonic anemometer are always present and have always the same format, no matter which of my own file type is used. These data are saved unprocessed in binary format as they arrive.

If there are additional instruments to be recorded, then this is indicated by the unique file type. Type **G** which I have not used so far, is a special one: if more than one instrument are combined, then each of the instrument will add one record of data per record received from the sonic anemometer in the order given by the naming. Thus, fog monitor data preceeds IRGA data.

A.2.1 Standard anemometer data

Data records contain 2-byte signed short integers for each component. The total size of one record depends on the number of analog channels that are switched on (see the settings in the header!). Without analog channels (that is, using a digital IRGA, for example), the sonic anemometer data contain 6 variables per record, which is 12 bytes (Table 4). With version 8.00 of `sonicreadHS` we introduced a change that we call version 2 WECOM3 format: the inclinometer data are no longer saved in bytes 9–12 as described in Table 4, but these 4 bytes are used differently as described in Table 5

Table 4: Base data from the sonic anemometer

Bytes	Component	Unit
1–2	u wind component	cm/s
3–4	v wind component	cm/s
5–6	w wind component	cm/s
7–8	virtual temperature measured via speed of sound	0.01 K
9–10	inclinometer, x-axis	0.01 degrees
11–12	inclinometer, y-axis	0.01 degrees

Table 5: Base data from the sonic anemometer saved in version 2 WECOM3 files (these files have bit 7 set in the `file_type` member of the header).

Bytes	Component	Unit
1–2	u wind component	cm/s
3–4	v wind component	cm/s
5–6	w wind component	cm/s
7–8	virtual temperature measured via speed of sound	0.01 K
9–10	StaA and StaD bytes[a]	binary copy from sonic data stream
11–12	inclinometer, x or y[b]	0.01 degrees

[a] StaA is status address (or record number) sent from the Gill sonic and StaD is the status data sent by the Gill sonic. The meaning of StaD depends on the value of StaA

[b] In records with StaA being an odd number the most recent x-axis component is saved, whereas in records with StaA being an even number the most recent y-axis component is saved

Note that the inclinometer data are treated as two analog channels which always are saved **after** all other analog channels in case there are any of them switched on.

The analog channels however differ in that the values saved in the file must be **multiplied by 0.61035283** to yield the raw voltages (see the Solent User Manual for details).

A.2.2 Additional data from digital instruments

If one or more digital instruments were operational then the base data are followed by at least two bytes of additional data per instrument per record (Table 6).

Table 6: Additional data per digital instrument per record; byte numbers refer to byte after the ending of the previous information, which may be either base data from that record or the data from the preceeding instrument.

Bytes	Content
1	byte count for the data from that instrument (at least 2 bytes)
2	instrument status code
3–n	data if any, depending on the number given in the first byte!

Thus, as an example for IRGA data: if we decided to record IRGA data, but the IRGA was not operational, then we just get two additional bytes per record from the anemometer. The first byte will tell us that there are two bytes to read, and the second byte gives us the reason why there are no more data (a status code).

If data are present, then the first byte must be greater than two, depending on the instrument and thus on the file type given in the header. For example, file type **I** has 16 bytes per record for the IRGA data if they were available.

A.3 FogMonitor data

The data files only contain the raw droplet counts for each of the 40 size bins of the FM-100. To know the diameter of each channel is a major issue, see Section A.3.10. A detailed investigation of the problem has been published by Spiegel et al. (2012).

A.3.1 Data Format and Flag Bits of the FogMonitor data

The data format of the messages of type `0x1bac0002` in the message queue is as follows.

All bytes are rearranged to follow the fully high-endian definition. All bytes are unsigned characters of 8 bit length, however, the numbers to be interpreted are either 8, 16 or 32 bit unsigned integer values.

The first byte gives the length of the message in bytes. Thus, after having read 1 byte and interpreted its content *n*, we should expect *n-1* more bytes to belong to this message.

The second byte is a status byte. The bits have the following meaning:

```
#define STATUS_OK                0000        /* no bits set      */
#define RESET_FLAG_SET           0001        /* bit 0 set        */
#define REJDOF_FLAG_SET          0002        /* bit 1 set        */
#define REJAVGTRANS_FLAG_SET     0004        /* bit 2 set        */
#define FIFOFULL_FLAG_SET        0010        /* bit 3 set        */
#define FM_DID_NOT_RESPOND       0020        /* bit 4 set        */
#define OLD_FM_DATA_USED         0040        /* bit 5 set        */
#define CONDENSED_DATA_MODE      0100        /* bit 6 set        */
#define MISSING_FM_DATA          0200        /* bit 7 set        */
```

Bits 0–4 and 6 are set by the `fm` program, while bits 5 and 7 are reserved for the sonic anemometer data aquisition program and should not be set by `fm`. In versions before 1.28 we had the following flag in place of `CONDENSED_DATA_MODE`:

```
#define CHECKSUM_ERROR           0100        /* bit 6 set        */
```

However, since data received with checksum errors are really bad data, we freed this flag and use it in a different way now (see Section A.3.9).

The full data structure of a message is

```
typedef struct
{
  unsigned char size;                          /* new! */
  unsigned char status;                        /* new! */
  unsigned char lwc[4];                        /* new unsigned long int */
  unsigned char time_us[4];                    /* new unsigned long int */
  unsigned char cabinChan[16];                 /* was 8 unsigned short int */
  unsigned char rejDOF[4];                     /* was unsigned long int */
  unsigned char rejAvgTrans[4];                /* was unsigned long int */
  unsigned char AvgTransit[2];                 /* was unsigned short int */
  unsigned char FIFOfull[2];                   /* was unsigned short int */
  unsigned char ADCoverrange[4];               /* was unsigned long int */
  unsigned char OPCchan[USED_OPCCHAN][4];      /* was unsigned long int */
}
FM_COOKEDDATA;
```

Note that we copy the analog channels data into this data structure because these are internal values which are necessary for data processing. Channel 3 contains the ambient temperature in the sample cell, channel 6 the static pressure and channel 7 the differential pressure. At least those are needed to compute liquid water content (LWC). For on-line control we now compute LWC and save it as an unsigned long integer because this format is more portable than any floating point binary format. We multiplied the original floating point LWC by a factor 1,000,000 rounded to the nearest integer value. The `time_us` member contains the time period in microseconds during which the data of this record were sampled. Because particle counts go up if the time interval between two data polls increases, we need to know this for postprocessing our data.

The last member of this structure contains the droplet-size fractioned channels. Their number `USED_OPCCHAN` is 40 at maximum and must be a multiple of 10. For example, if we used 20 channels, then the size of the message would be $20 \times 4 + 20 = 100$ bytes. Thus if byte 0 contains the number 100, we know that there are 20 channels of data present.

To interpret the numbers we can proceed as follows. For 16 bit (2 byte) integers, we multiply the high byte (the one that is nearest to the size byte) by 256 and add the contents of the next byte. For example, if we defined a variable `data` of type `FM_COOKEDDATA`, we can get the short unsigned integer variable `fifo` by

```
fifo = data.FIFOfull[0]*256+data.FIFOfull[1];
```

A.3.2 Status Flag Details of the FogMonitor data

The status byte has 8 bits, each of which has its special meaning according to the definition given in Section A.3.1. A bit value of 0 means that the flag is unset, a bit value of 1 means that the flag is set.

The definitions for the flag positions given in Section A.3.1 are octal values. For example, the value 0004 is bit 2 and 0010 is bit 3.

A.3.3 Reset Flag

The reset flag position is defined by `RESET_FLAG_SET`. It is **not** just the copy of the reset flag received from the FogMonitor, because our `fm` program will detect when the FogMonitor sends data with the reset flag high. If this is the case, it will resend the setup data structure to the FogMonitor and see, if the reset flag is low in the next data set. If not, it will continue to try to setup the FogMonitor. Thus, the data with the reset flag high are not forwarded to the message queues. Instead, we label the first data block with valid data that was received **after** we sent the setup data structure to the FogMonitor with a high reset flag in our setup byte. In this way we will know at the time of postprocessing, that this is the first valid data block after a restart or reconfiguration of the FogMonitor.

A.3.4 RejDOF Flag

The RejDOF flag position is defined by `REJDOF_FLAG_SET`. The data block contains a member `rejDOF` which is a counter for the number of particles that were rejected because the signal quality is lower than the annulus. If this value is above a certain threshold, we will raise the RejDOF flag in our status byte,

because it may indicate improper operation of the FogMonitor. At the moment this threshold is set zero, but depending on our experience to be gained it may be set to a higher value later.

A.3.5 RejAvgTrans Flag

The RejAvgTrans flag position is defined by `REJAVGTRANS_FLAG_SET`. The data block contains a member `rejAvgTrans` which is a counter for particles that were rejected because they did not meet the criteria for transit time. If this value is above a certain threshold, we will raise the RejAvgTrans flag in our status byte, because it may indicate improper operation of the FogMonitor. At the moment this threshold is set zero, but depending on our experience to be gained it may be set to a higher value later.

A.3.6 FIFOfull Flag

The FIFOfull flag position is defined by `FIFOFULL_FLAG_SET`. The data block contains a member `FIFOfull` which is a counter for the number of times where the FIFOs were full and were flushed. Seemingly this might happen in very dense fog. If this value is above a certain threshold, we will raise the FIFOfull flag in our status byte, because it may indicate improper operation of the FogMonitor. At the moment this threshold is set zero, but depending on our experience to be gained it may be set to a higher value later.

A.3.7 No Response Flag

The No-response flag position is defined by `FM_DID_NOT_RESPOND`. If this flag is set, this means that we did not receive a complete data block from the FogMonitor, or we did not get any response at all. Such data will be ignored and not sent to the message queues. The principle is the following: if we poll data from the FogMonitor but do not get the appropriate number of bytes back (or no response at all), after a certain wait delay (currently set to 65 μs), our `fm` program will return and make one additional attempt to read the remaining amount of data. If the data block is still not complete, the No-response flag is set high. The `fm` program will now try to poll the next data block which normally is ok. Our experience shows that this kind of data losses occurrs at some load level when one tries to do multiple things on the laptop at the same time. During normal operation, there should be very few data blocks that are missed.

A.3.8 Checksum error flag

As of version 1.24 we do no longer deliver data blocks with checksum errors to the message queue **F**, only to queue **G**. This means, that such data will not be saved on disk.

We'll have to gain experience on how frequently this occurs and whether we need to re-send the setup information if a checksum error occurs. In Version 1.24 we treat them like missed data blocks or data blocks which were incomplete (not the full length).

In Version 1.28 we even got rid of this flag and reused its magic number for labeling condensed-mode data records (see Section A.3.9).

We keep control over what's going on by updating the log file periodically (see Section 7).

A.3.9 Condensed Data Mode

To lessen the amount of data to be saved on disk, especially during periods where there is no fog, we introduced two condensed data modes in Verison 1.28:

- Mode 1: if all droplet counters of all size fractions are zero, we only save the base data without the channels containing the droplet counts. In this mode the typical block size is 42 bytes independent of the number of channels used in the FogMonitor. This condenses the FogMonitor data by 65% compared to the standard mode without condensation.
- Mode 2: if some of the droplet counters were greater than zero, but none of them exceeded 255, we saved the counters in 1-byte integers instead of the standard 4-byte integers. This means that the block size is 42 bytes plus the number of channels, thus 62 bytes if we used 20 channels. Compared to the non-condensed size of 122 bytes (42 bytes of base data plus 20 times 4 bytes of droplet counts) this is still a condensation by 49%.

If the `CONDENSED_DATA_MODE` is not set then the data block size is either 2 bytes if no FogMonitor data is available, or it is 42 bytes plus 4 time the number of droplet size fractions (channels) in use. For example, with 20 channels active this would be 122 bytes.

A.3.10 How to know the droplet size definitions of each FM-100 channel?

The boundaries for each channel of the FM-100 are defined in `fmsetup.c` of the `fm` program. Since we have a collection of possible settings there it may not be perfectly clear what we actually used. Thus, we should always save the information of the settings that we used in a separate file that allows us to compute the droplet sizes later.

This can be done with a command line option to `fm`: For better control about what we are doing and what we will be doing a new option was introduced in version 1.34 of `fm`, the `-s` option.

When starting `fm` with this option there is no need to have a connection to the instrument, it works also offline. It saves the current setup string into a file with the name defined in `fm.h` by SETUPSTRING_FILE. Currently we use the name "setup.giub". Note that on MacOS X this file for some unknown reason only has the executable flag set and is not readable until we change the permissions with

```
chmod a+r setup.giub
```

This string can then be decoded with a tool in the TOOLS directory of `fm`, `decodesetup` (compile it first in the TOOLS directory of `fm` using the command `make decodesetup`):

```
decodesetup setup.giub
```

The output is similar to this:

```
ESC = 0x1b
COMMAND_NR = 0x01
threshold = 20
transRej = 0x00
chanCnt = 40
dofRej = 0x01
flags = 0x02
avgTransWeight = 5
attAccept = 95
divisorFlag = 0x00
ct_method = 0x01
  channel[00] threshold   17
  channel[01] threshold   42
  channel[02] threshold   92
  channel[03] threshold  149
  channel[04] threshold  192
  channel[05] threshold  236
  channel[06] threshold  282
  channel[07] threshold  334
  channel[08] threshold  369
  channel[09] threshold  403
  channel[10] threshold  437
  channel[11] threshold  526
  channel[12] threshold  601
  channel[13] threshold  681
  channel[14] threshold  764
  channel[15] threshold  852
  channel[16] threshold  943
  channel[17] threshold 1038
  channel[18] threshold 1152
  channel[19] threshold 1252
  channel[20] threshold 1356
  channel[21] threshold 1463
  channel[22] threshold 1572
  channel[23] threshold 1685
  channel[24] threshold 1779
  channel[25] threshold 1908
  channel[26] threshold 2041
```

```
  channel[27] threshold 2178
  channel[28] threshold 2319
  channel[29] threshold 2464
  channel[30] threshold 2612
  channel[31] threshold 2764
  channel[32] threshold 2921
  channel[33] threshold 3080
  channel[34] threshold 3244
  channel[35] threshold 3411
  channel[36] threshold 3582
  channel[37] threshold 3730
  channel[38] threshold 3911
  channel[39] threshold 4096
cksum = 5295
```

Although our software is generally made in a way that does not need any DOS or Windows utilities from the manufacturer of the FM-100, it is not sufficiently described in the manual of the FM-100 how to compute the droplet diameters from the above thresholds. We therefore used their PACS software to just get the information of what the geometric mean droplet size of each channel actually is. This information is also used in `fmsetup.c` to compute LWC on the fly.

If we want to do our own analysis with the raw data we have to be able to assign a droplet size for each of the channels. If we have not done this before, we can enter all the threshold values above to the PACS software and then yield the droplet sizes that we could use. For our standard setting these would be the aerodynamic mean diameters for each channel (in R syntax):

```
dropsize <- c("1.500","1.93649","2.95804","3.96863","4.97494","5.97913","6.98212",
  "7.98436","8.98610","9.98749","10.98863","11.98958","12.99038","13.99107",
  "14.99166","15.99219","16.99265","17.99305","18.99342","19.99375","20.99405",
  "22.23736","23.73815","25.23886","26.73948","28.24004","29.74054","31.24100",
  "32.74141", "34.24179","35.74213","37.24245","38.74274","40.24301","41.74326",
  "43.24350","44.74371","46.24392","47.74411","49.24429")
```

A.4 IRGA data

A.4.1 Licor 7500

The only specialty here is that we used 3-byte integers for the H_2O and CO_2 data arriving from the LiCOR 7500 since there are no additional significant bits necessary than the 24 bits provided in this way. However, when reading in binary data you should be aware that you need to read the data byte-by-byte, because other data structures may be aligned with even numbers of bytes (which normally confuses beginners who are not aware that empty bytes are inserted in such reads depending on platform and programming language).

Table 7: IRGA data saved from the LiCOR 7500 open path instruments.

Bytes	Component	Unit	Offset
1	size of IRGA data block	bytes	
2	status of IRGA data aquisition	code, Table 8	
3	status byte from the LiCOR 7500 IRGA	code	
4–6	H2O concentration	0.001 mmol m^{-3}	
7–9	CO2 concentration	0.0001 mmol m^{-3}	
10–11	IRGA temperature	0.01 K	100 K
12–13	IRGA pressure	10 Pa	
14–16	IRGA cooler voltage	0.0001 Volts	

As an example, the H_2O data are computed from bytes B_4, B_5, and B_6 in the following way:

$$H_2O \text{ concentration} = 0.001 \cdot (65536B_4 + 256B_5 + B_6) \ ,$$

which yields the values in mmol m^{-3}.

Status codes of the data aquisition are given in Table 8, where the values in the data file are a binary OR combination of the status bits. Under normal operation, only codes 0000 and 0040 octal should occur. A special note follows on code 0040.

Table 8: Codes of the IRGA data aquisition

Bits set	Binary	Octal	Description
none	00000000	0000	Status OK, no problems
5	00010000	0020	IRGA did not respond
6	00100000	0040	Status OK, old data used
8	10000000	0200	not OK, IRGA data are missing

Code 0040: This is the code to show that we are either undersampling the IRGA, or—if the IRGA should deliver data at the same nominal rate as the sonic anemometer—that the IRGA data are laging the sonic data by a certain time period. By default I have set up the `MAX_IRGA_REPLICATION` definition in the source code to allow for 5 consecutive replication of old IRGA data records. That means that if we run the sonic at 100 Hertz but the IRGA at 20 Hertz, the 20-Hertz data will be copied 5 times into the merged binary file without any additional error code. But if for example the IRGA happens to fail, after 5 copies we will get code 0200 and the IRGA data structure in the binary file will immediately be reduced to the 2 bytes which are always present in the file.

Under normal operation, when both the sonic and the IRGA are running at 20 Hertz, there is an undulation between periods with code 0000 and periods with 0040 where this only indicates that in the first case the IRGA data have already arrived in our data queue when we try to merge it with the sonic data, and in the latter case we have not received the new data yet and thus will use the old record (the new record will thus arrive shortly after we have merged the data and the next time. This timing issue is normal with asynchronic data aquisition and has no implication on data quality because we compute the time lag per averaging interval anyway.

A.4.2 Licor 7000

We used the same concept when we started to support the Licor 7000 closed path instrument. However, in order to be able to keep the Licor 7500 data apart from the Licor 7000 data, we use a record size that is only 15 bytes in the case of the Licor 7000 (it is 16 bytes for the Licor 7500). Table 9 shows the details.

Table 9: IRGA data saved from the LiCOR 7000 closed path instruments.

Bytes	Component	Unit	Offset
1	size of IRGA data block	bytes	
2	status of IRGA data aquisition	code, Table 8	
3	Diag byte from the LiCOR 7000 IRGA	code	
4–6	H2O concentration	0.001 ppt	
7–9	CO2 concentration	0.0001 ppm	
10–11	IRGA cell temperature	0.01 K	100 K
12–13	IRGA cell pressure	10 Pa	
14–15	internal pump voltage	0.0001 Volts	

A.4.3 Licor 7200

Support for the Licor 7200 was introduced in summer 2012 in version 7.02, but in contrast to the Licor 7500 and Licor 7000 we switched to TCP/IP communication for the Licor 7200 (see the `licor` program).

In July 2017 we slightly changed the data format in `sonicreadHS` version 8.00 or higher: here the Licor 7200 status information is saved as 2 bytes whereas with older versions of `sonicreadHS` this status information was only 1 byte (a change that reflects the change in Licor firmware of the instrument). Note that the change is made in `sonicreadHS`, not in `licor`, but this affects the position of the variables in the binary data structure shown below.

There was one problem that could not be solved at the time the data acquisition was written and which affects all data collected until this may (or may not) be solved in a future version: every 20th record or

so a flow rate of 0.0 L min^{-1} is reported which leads to an erroneous CO_2 and H_2O concentration in the primary variable(the dry mole fraction) whereas the mole density is OK. In the code I simply reduced the screening boundaries in `sonicavg.h` (upper margin is now 1500 instead of 5000 for CO_2 and H_2O, respectively).

Another issue is that version 7.02 used at the Toolik Wetland site in 2012 stored an erroneous version and date information (in WESAT3 header information). When checking the new Davos data processing we will have to check for this in the WECOM3 format as well!

The data are stored as we would store those of the Licor 7500, but the byes have a different assignment. Only the first 3 bytes are the same as for the 7500, the others are as decribed below and in Table 10. Important: Records from the Licor 7200 are saved with 25 or 26 bytes binary, those of the Licor 7500 have 16 bytes and those of the Licor 7000 have 15 bytes. Note that we have 25 bytes from the Licor 7200 if we recorded with `sonicreadHS` version up to 7.09 but 26 bytes since version 8.00. **The structure below shows the 25-byte variant**—for the 26 byte variant the only difference is that IRGADATA_LICOR has two bytes. In my converter I did not change anything with the structure, but if IRGADATA_SIZE shows 26, then I read the extra byte after IRGADATA_LICOR; after having read this extra byte of the 2-byte status information, then all other variables are interpreted correctly.

```
#define IRGADATA_SIZE            0  /* should be 0 -> first byte of record          */
#define IRGADATA_STATUS          1  /* should be 1 -> second byte of record         */
#define IRGADATA_LICOR           2  /* this is the Licor 7500 status byte           */
                                    /* or the Licor 7000 Diag number                */
                                    /* if 2 bytes from the Licor 7200 are saved,    */
                                    /* then we have to read an extra byte here!     */
#define IRGA7200DATA_H2OMFD      3  /* bytes 3-5:   H2O dry mole fraction mmol/mol  */
#define IRGA7200DATA_CO2MFD      6  /* bytes 6-8:   CO2 dry mole fraction umol/mol  */
#define IRGA7200DATA_H2OD        9  /* bytes 9-11:  H2O density mmol/m3             */
#define IRGA7200DATA_CO2D       12  /* bytes 12-14: CO2 density mmol/m3             */
#define IRGA7200DATA_TEMP       15  /* bytes 15-16: IRGA temperature                */
#define IRGA7200DATA_PRESS      17  /* bytes 17-18: IRGA pressure                   */
#define IRGA7200DATA_APRESS     19  /* bytes 19-20: atmospheric pressure            */
#define IRGA7200DATA_COOLER     21  /* bytes 21-22: IRGA cooler voltage             */
#define IRGA7200DATA_FLOWRATE   23  /* bytes 23-24: IRGA volumetric flow rate       */
#define IRGA7200DATA_RECLEN     25  /* total length of binary IRGA 7200 record      */
```

Table 10: IRGA data saved from the LiCOR 7200 enclosed path instruments. In blue the position is shown for 26-byte records, whereas the black byte positions are showing the older 25-byte records.

Bytes	Bytes	Component	Unit	Offset
1	1	size of IRGA data block	bytes	
2	2	status of IRGA data aquisition	code, Table 8	
3	3	Diag byte from the LiCOR 7200 IRGA	code	
5–7	4–6	H2O dry mole fraction	umol/mol	
8–10	7–9	CO2 dry mole fraction	umol/mol	
11–13	10–12	H2O concentration	0.001 mmol m^{-3}	
14–16	13–15	CO2 concentration	0.0001 mmol m^{-3}	
17–18	16–17	IRGA temperature	0.01 K	100 K
19–20	18–19	IRGA pressure	10? Pa	
21–22	20–21	atmospheric pressure	10? Pa	
23–24	22–23	IRGA cooler voltage	0.0001 Volts	
25–26	24–25	IRGA volumetric flow rate	XXXX	

In the first few files the header has this error: TO BE WRITTEN. The header information defines them as file type I, if only one IRGA is used and this one IRGA is a Licor 7200.

Well, currently I don't see where date and header info should be wrong, maybe this was a version before I used it productively at Toolik Wetland – check the log files and then rewrite this text!

A.4.4 Raw data errors of the Licor 7200

The Licor 7200 instrument used at Toolik Wetland has the following low-level bug that needs to be taken care during data processing:

In more or less regular intervals a record arrives that has a temperature of 0.0 or –1.0, a cell pressure of 0.0 or 20.0 and a cooler voltage of 0.0. Also the AGC values seems to be 0. Since these values are not all at the end of the data record sent by the Licor 7200 it is very unlikely that this has to do with the data acquisition. I suspect a bug in the system where the A/D system that measures these extra variables is either behind the schedule for collecting internal variables and then the system simply sends zeros. We receive the data in the following order:

```
(Data (DiagVal 8184)(CO2D 20.292)(H2OD 521.757)(Temp 26.9615)(Pres 91.4931)(APres 91.5545)
(Cooler 2.109)(CO2MFd 561.38)(H2OMFd 14.4345)(DewPt 10.8279)(VolFlowRate 0))
```

as of early 2013 we only recorded the lower 8 bits of the new 16-bit diagnostic data structure, but it turns out that for tracking down such instrument issues it would be better to have both data bytes. Should be modified in a follow-up version of the irga data acquisition program and sonicreadHS.

For now we have the feeling that the CO2D and H2OD data, the APress and VolFlowRate values are OK and hence we should be able to deduce correct CO2MFd and H2OMFd values. But since these errors are not perfectly synchroneous with the same variables (when it actually happens) I decided to simply mark these values by MISSINGVALUES since we have 96.9% of good data in the test data file from 2012-07-28 that I inspected in detail.

Although I initially wrongly interpreted the issue to be the reporting of a zero flow rate by the Licor 7200 in such cases, my detailed investigation showed that in the subset of data I analyzed the flow rate is always correct, as is the ambient pressure and the volumentric CO2 and H2O concentration. So if you come across some notes saying the flow rate was zero this is an error (I forgot that my tests of which I report the data format of the Licor 7200 was done without the pump running, hence the 0 flowrate – the algorithm correctly converts the concentrations even at zero flow rate).

A.5 Secondary IRGA data

Starting with version 7.03 we allow to record IRGA data also from a secondary instrument that we specify with the `-I#` command line option. The data are saved in exactly the same way as described in Section A.4, and the data block from the secondary IRGA immediately follows the data block from the primary IRGA.

To distinguish files with two IRGA data blocks from those with only one, we use the `file_type` **K** instead of **I**. This means that we only set bit 1 in addition to what we have for type **I** which should make it simple to treat these files like type **I** files but with an extra IRGA data block.

A.6 Extension data (QCL, LGR)

For further developments we introduced the concept of extensions, which so far were an Aerodyne QCL system and since May 2008 a Los Gatos Research (LGR) methane analyzer. In May 2009 we extended this to the new Los Gatos Research FGGA (Fast Greenhouse-Gas Analyzer). To mark our data files for such extensions we had to sneak in some information into the header as described in Table 3. Additionally, a variant of each extension version is foreseen and saved in each extension record.

Extension data are always appended to the previous records in the sequence standard data – fog monitor data – IRGA data – extension data. Each of the additions to the standard records has at least two data bytes that must be read:

Byte 0 contains the length of the record (thus this value must be at least 2, at max 255); this thus limits the possibility for each extension to 253 bytes plus the two mandatory bytes.

Byte 1 contains the status information intermingled with the extension variant information. The most significant bits of this byte (bits 4–7) contain the variant information, whereas the least significant bits (bits 0–3) contain status information. The status codes are found in `extdata.h`. We should carefully keep our Table 3 updated such that we can always find out what these extension data represent.

A.6.1 Data from a second QC-TILDAS

In `sonicreadHS` V. 7.09 the `-e2.2` option was introduced which receives and saves QCL data from two systems, with the first running as `qclread -s0`and the second running as `qclread -s1` plus command-line long options to specify the components and their order in the respective datastream. For simplicity, a species-specific conversion is programmed in (see Table 11).

Table 11: Multipliers and offsets in extension version 2 variant 2 data. `sonicreadHS`takes the values as they arrive from `qclread`, thus the conversion is defined there (look at `getopt_long` in `qcl.c`). Data are saved as 4-byte integers.

qclread Version	sonicreadHS Version	Known Application	Component	Multiplier	Offset
2.04	7.09	Lutz Merbold, Mazingira project	ch4	10000	0
		November 2016–	n2o	10000	0
			co2	100	0
			h2o	1	0
			no2	10000	0
			nh3	10000	0
			cellT	10000	0
			cellP	10000	0

A.6.2 Los Gatos Research Data

Currently we have two variants of data, one for the LGR Fast Methane Analyzer (FMA, Table 12), and one for the Fast Greenhouse-Gas Analyzer (FGGA, Table 13). They can easily be distingushed either from their block size (byte 1 of the data block), or from the extension version information scrambled into the status byte (byte 2), which is explained in Table 15.

Table 12: LGR FMA Data (extension version 1, variant 3).

Bytes	Component	Unit
1	size of LGR data block	bytes – should be 15 (with data) or 2 (no data)
2	status of LGR data aquisition	code, see Table 15
3–6	CH4 concentration	ppb $\times$ 10,000
7–8	LGR cell pressure	Torr $\times$ 100
9–10	LGR cell temperature	°C $\times$ 100
11–14	LGR mirror ringdown time	$\mu s \times$ 1,000,000
15	Calibration flag	seems to be 0 with the FMA

Table 13: LGR FGGA Data (extension version 1, variant 4). This format uses the same format as the LGR FMA Data (see Table 12) to which we added the three additional variables.

Bytes	Component	Unit
1	size of LGR data block	bytes – should be 27 (with data) or 2 (no data)
2	status of LGR data aquisition	code, see Table 15
3–6	CH4 concentration	ppb $\times$ 10,000
7–8	LGR cell pressure	Torr $\times$ 100
9–10	LGR cell temperature	°C $\times$ 100
11–14	LGR mirror ringdown time #1	$\mu s \times$ 1,000,000
15	Calibration flag	seems to be 2 with the FGGA
16–19	CO2 concentration	ppm $\times$ 10,000
20–23	H2O concentration	ppm $\times$ 10,000
24–27	LGR mirror ringdown time #2	$\mu s \times$ 1,000,000

In contrast to the IRGA data we moved the status bits in our data records to the 4 least significant bits (see Table 15) and use the 4 most significant bits (bits 5–8) to save the extension **variant** here. This is expressed with xxxx and yy in Table 15. Thus, for decoding, the extension variant should always be the same in all records, irrespective of its length, whereas in the current implementation a specific extension and variant can only have two possible lengths, either 2 (which is the minimum with the byte specifying the length of the record and the second byte being the status information).

It is possible to find out which records were really measured and which ones were just replicated because no new data record was available when the newest sonic anemometer data record was read. This

Table 14: LGR N2O/CH4/H2O Data (extension version 1, variant 7) introduced with `sonicreadHS` version 8.04 and lgrread version 2.00.

Bytes	Component	Unit
1	size of LGR data block	bytes – should be 33 (with data) or 2 (no data)
2	status of LGR data aquisition	code, see Table 15
3–6	CH4 dry mole fraction	ppm × 10,000,000
7–10	N2O dry mole fraction	ppm × 10,000,000
11–14	H2O concentration	ppm × 10,000
15–18	CH4 concentration in moist air	ppm × 10,000,000
19–22	N2O concentration in moist air	ppm × 10,000,000
23–24	LGR cell pressure	Torr × 100
25–26	LGR cell temperature	°C × 100
27–28	LGR ambient temperature	°C × 100
29–32	LGR mirror ringdown time	μs × 1,000,000
33	Fit flag	an integer, 0–3

Table 15: Codes of the LGR data aquisition

Bits set	Binary	Octal	Description
none	xxxx0000	yy00	Status OK, no problems
1	xxxx0001	yy01	Old LGR record replicated
2	xxxx0010	yy02	LGR did not respond (this is never used)
3	xxxx0100	yy10	not OK, LGR data are missing

is shown with bit 1 high in the status byte ("old LGR record replicated"). We hard-coded the maximum number of replications in the source code. See `MAX_LGR_REPLICATION` in `extdata.h`.

B Binary WESAT3 Raw Data Format

B.1 Header

Each data file starts with a 38 byte header of the following format:

```
typedef struct /* the new WESAT3 header size is 38 bytes */
{
char format[6]; /* will hold the characters WESAT3 */
unsigned char file_version; /* in case the WESAT3 header needs changes */
unsigned char file_type;
unsigned char firmware_version[6];
unsigned char serial_number[4];
unsigned char execution_para;
unsigned char trigger_src;
unsigned char analog_range;
unsigned char data_status;
unsigned char terminal_mode;
unsigned char prompt;
char hostname[10];
unsigned char file_create_time[4];
} HEADERTYPECSAT3;
```

These are the elements in this header:

- **format**: this contains the constant string WESAT3 that allows for easy detection of the data file format in the future.
- **file_version**: this is a version number (0–255) that will allow us to keep track of any substantial changes that need to be labeled with a new version number. With the introduction of this file format

we started with version 0 in February 2007. This number will only increase if it is necessary. Note that many variants of how data are arranged in these files are clearly determined otherwise (see later).

- **file_type**: exactly the same as for WECOM3 files (see Section A.1) but it should be understood that numbers 0–3 should not occur in this field since these numbers specify the RCOM3 formats from Solent sonics, and we never write WESAT3 files if we run Solent sonics. Thus, having a CSAT3 and using WESAT3 output format basically excludes these numbers from being used here. The character codes we expect here are the following. Type **A** is specifying sonic-only data (no extra instruments), saved as decimal character 65; type **F**, specifying sonic plus FogMonitor is decimal character 70; type **G**, sonic plus FogMonitor plus IRGA, is decimal character 71; type **I**, sonic plus IRGA, is decimal character 73 (corresponding to the position of the capital letters in the ASCII character set). All other characters are a binary **OR** combination of these basic types with the extension version and variant (e.g. when adding QCL data or any other data from an extension instrument).

- **firmware_version**: this 6-character field is prepared to hold the ASCII representation of the CSAT3 version information received via the status information. As of Version 5.02 we still have not found the time to implement this and thus this field is currently 6 bytes of null characters (0x00).

- **serial_number**: serial number of the CSAT3 sonic used in 4-byte ASCII representation as received via the status information from the sonic.

- **execution_para**: the execution parameter setting received via the status information from the sonic. This information defines the sampling rate (see CSAT3 manual for the possibilities there are). **h** is our standard in Panama and means internal 60-Hertz measurements that were averaged by the sonic to 20 Hertz output.

- **trigger_src**: trigger source information as received via the status information from the sonic, should be **0**.

- **analog_range**: see CSAT3 manual, no relevance for our data files.

- **data_status**: CSAT3 data status at startup, should be "good data" (that is **0**).

- **terminal_mode**: CSAT3 terminal mode status, should be **T**.

- **prompt**: CSAT3 prompting mode status, should be **U** for "unprompted mode".

- **hostname**: in this 10-byte field we save the host name of the computer that ran the data acquisition. If the hostname is more than 10 characters then it will be truncated to exactly 10 characters. In our set-ups we give the computer that does the data acquisition the unique name of the site, thus this field allows us to know where the data came from in case there are duplicate filenames (by default we do not save the site information in the file names).

- **file_create_time**: exactly the same as for WECOM3 files (see Section A.1). In my variant of this binary format I did use the time stamp in local time, that is, this 4-byte long integer gives the seconds since 00:00:00 local time, 1 January 1970. To my understanding, the RCOM3 format is using time in UTC, thus depends on the time zone you have selected on your computer while doing the data aquisition. I have also used the little endian encoding also for the time stamp as in RCOM3, but be aware that all other data are saved in big endian format!

B.2 Data

After this header the binary data start with the first sonic data record, followed by the data from additional digital instruments in the order: FogMonitor, IRGA, Extensions. If we use different types of IRGAs we will make sure that they are saved with a different number of bytes (which is anyway the most realistic case because each IRGA has special variables that the other does not have) and thus it will be possible to determine the type from the number of bytes of these records.

The same applies to the extension data, but here it will most likely not be possible to keep track in the same way as with the IRGA since the extensions can also be experimental (not necessarily operational as with FogMonitor and IRGA), thus the same instrument may be saved with different numbers of bytes

depending on short-term decisions. Thus, we use the possibility to specify an **extension version** and an **extension variant**. The idea is that the versions are only changed if absolutely necessary since we only have versions 1–7 available unless we change the header format of our files. As of `sonicreadHS` version 5.02 we only have extension version 1 implemented and so far we only used version 1, variant 0 for QCL data from EMPA (and accidentally also variant 2, which however is no problem to be re-used – it was only a bug in the first week of operation when this concept was introduced where we saved IRGA data as duplicates in the place where QCL data should have been).

For future developments: I suggest that version 1 is assigned to all the experimental stuff that Werner Eugster does (currently QCL, but it could be more), and version 2 will be used by similar projects of other people (in this way they can keep track of the variants within that specific version themselves). Table 3 tries to keep track of all known developments.

B.2.1 Standard anemometer data

Data records from the CSAT3 sonics are 12 bytes long, consisting of 10 bytes of data (2-byte short integers) plus two delimiters that we also save in the files. The CSAT3 have no analog inputs and thus the record size is exactly the same for all variants. The records are saved in the format they arrive and thus the detailed description in the CSAT3 manual can be used to decode the data. Table 16).

Table 16: Base data from the CSAT3 sonic anemometer

Bytes	Component	Unit
1–2	u wind component	depends on range
3–4	v wind component	depends on range
5–6	w wind component	depends on range
7–8	speed of sound (difference to 340.0 m/s)	mm/s
9–10	diagnostic word (status, ranges, counter)	—
11–12	delimiters: 0x55 followed by 0xaa	—

The CSAT3 uses a little endian byte order, that is, in a 2-byte word we find the most significant byte not first, but last in the row of bytes (this is typical for Intel-type processors, whereas Sun Sparc and others use the high-endian byte order). During data acquisition we never do any rearrangement of byte orders. But it should be kept in mind that if problems occur when decoding binary files, it could be related to byte order issues, although we tried hard to get a binary format that does not depend on the architecture of the computer hardware of the data acquisition system.

B.2.2 IRGA, QCL and LGR data

If additional data are recorded in combination with the CSAT3 data stream, then the additional records with IRGA, QCL and/or LGR data are appended to the binary data records from the CSAT3 in the same way as in the WECOM3 data format. See details under Section A.4, A.5, A.6.

C Hints and Suggestions

When decoding raw data there are a few tricky issues to be solved:

- in general we use the concept to save the higher byte first, followed by the lower significant byte(s) in the data stream; however, in the Solent header a mixture is found where the time is saved in the reverse notation
- an odd number of bytes of the header can be tricky to read with some software
- the concept of only saving 2 bytes if no data from an instrument we actually want to record are available gives trouble with some software

The suggestion is to use `ethconvert` to translate binary raw data files to ASCII files if needed. I typically use this in combination with the statistics software **R**, using a pipe() to read in data output by `ethconvert`. This also solves the issue with intermediate (huge) ASCII files: they are not needed since one can directly work with raw data files.

C.1 How to get a CSAT3 up and running

There are some special issues one must know when using a Campbell CSAT3 with this software:

- What to do when the CSAT3 comes new or repaired from Campbell or does not work due to other reasons?

What to do when the CSAT3 comes new or repaired from Campbell or does not work due to other reasons?

Campbell has this interesting concept that a CSAT3 does not do what it's supposed to do unless you tell it that it should behave like a measuring device. Here are some hints. The easiest way (if it works) is to activate the sonic manually. To do this, hook up the serial port to one of your computer's serial ports, power on the CSAT3, then open two Linux terminal shells. In one you key in a command to read from the sonic, in the other you key in commands to be sent to the sonic. Since it works at 9600 baud, you do not have to set any specific port settings, just try with the defaults. If your serial port is known as `/dev/ttyM0` under Linux (that would be the first serial port, COM1, on a Moxa embedded computer), then key in

```
cat /dev/sonic
```

in your first terminal shell. Nothing will be shown except the command, but just be patient. Then go to your second terminal shell. Here you key in the command

```
echo "S\r" >> /dev/sonic
```

Now watch the text appearing in the first terminal mode. If it looks like this example: `0g02D1836U`, then we are in good shape; we now can switch to automatic mode by keying in the following in the second terminal window:

```
echo "&\r" >> /dev/sonic
```

Now data should arrive in continuous mode in the first terminal window. If the data are not arriving at a frequency of 2 Hz or more, then our software may give you an error message and stop. In this case, you must increase the output frequency by sending another command,

```
echo "&\r" >> /dev/sonic
```

If you think that your sonic is not sending at a high frequency (e.g. you only see a group of question marks every second), then also send the command to put it to 10 Hz before trying out `sonicreadHS`:

```
echo "A9\r" >> /dev/sonic
```

Now you should see a set of binary data arriving quite quickly. Note that unreadable characters are shown as question marks, and since backspace etc. can be in the binary data, sometimes you see a step back in the arriving data, which is just OK – `sonicreadHS` will configure the sonic anemometer to its correct settings, but to be able to do so it must find the sonic, and the assumptions in the software are (a) that the sonic is sending data continuously (thus, cannot be in polled mode or triggered mode), and (b) that it is sending at a frequency which is higher than 2 Hz typically.

Finish your work by pressing Ctrl+C in the first terminal window where you see your data arriving. Then you can test whether our acquisition program works by using the verbose option at startup, that is

```
sonicreadHS -v -M CSAT3 -o
```

For this to work there **must** be a device with the name `/dev/sonic` otherwise you will get a message tellling you to create this device (or make a symbolic link).

For troubleshooting purposes you find the output of a correct startup sequence in the following (most of this output will be suppressed if the -v switch is not used):

```
root@Moxa:/var/hda/data/Moxa# /var/hda/progs/sonicreadHS -v -M CSAT3 -o
Option M with optarg CSAT3 detected
Using Campbell CSAT3 protocol for sonic anemometer
----------------------------------------
Switching CSAT3 delimiter mode OFF
----------------------------------------
Sonic protocol number used: 3
Highest priority on system is 99, lowest is 1
We try to get priority 1

Alarm signal received.
COMMENT     This is a sample sonicrc file by Werner Eugster
COMMENT     which is used for the CarboEuropeIP data aquisition systems
DATAPATH    ./
COMMENT     the old sonic anemometer has 5, the new HS sonic 6 analog channels
ANALOGS     0
COMMENT     MODE is ignored; leave it with a 1
MODE        1
COMMENT     the old sonic anemometer works with 4800, 9600 or 19200
COMMENT     the new HS sonic anemometer works with 9600, 19200 or 38400
BAUDRATE    4800
COMMENT     CHANGEFILE is an integer; after ... hours a new file will be opened
CHANGEFILE  6
COMMENT     FILECODE is one character that will appear in the filename ext.
FILECODE    b
LOGFILE     Moxa.log

COMMENT     below are additional settings for the new Solent HS sonic
COMMENT     AVERAGE 1 is 100 Hz, AVERAGE 5 is 20 Hz, AVERAGE 10 is 10 Hz
AVERAGE     5
COMMENT     define where IRGA data should be tanken from for averaging
COMMENT     valid options are: ANALOG, DIGITAL, BOTH or NONE
IRGA        DIGITAL
COMMENT     IRGA 2nd order conversion from millivolts to concentration
COMMENT     introduced in version 3.01; conc = a0 + a1*x + a2*x^2
COMMENT     format: three floating point numbers for a0, a1, and a2
COMMENT     only used for analog channel measurements
IRGA_CO2    0.0 1.0 0.0
IRGA_H2O    0.0 1.0 0.0
message queue 65538 installed.
opened serial port /dev/sonic
   If program hangs when trying a baud rate then you must
   unplug the power supply to the sonic and plug it in again.
   If you use an old sonic you MUST use the -R2 switch at startup
   otherwise the program will wait infinitely shortly after
   If you use a CSAT3 sonic you MUST use the -MCSAT3 switch at startup

Trying baud rate 15...

Synchronizing data WITHOUT Delimiter
TEST Pt 1
Pass 0
 (bytesavailable=0) --fail++ (result=1) . TEST Pt 2
INFO: Now trying undelimited mode (Test Pt 3)
bytesavailable=100
i=0: 37 41 42 47
i=1: 63 63 63 63
i=2: 55 60 52 47
i=3: 0 0 0 0
i=4: 6 6 3 4
i=5: 63 63 63 63
i=6: 14 41 1 8
i=7: 22 22 22 22
```

```
i=8: 11 12 13 14
i=9: 15 15 15 15
INFO: FOUND block structure at offset 8

Sonic found
CSAT3 sampling rate g

Synchronizing data WITHOUT Delimiter
TEST Pt 1
Pass 1
 (bytesavailable=0) --fail++ (result=1) . TEST Pt 2
INFO: Now trying undelimited mode (Test Pt 3)
bytesavailable=100
i=0: 36 38 44 53
i=1: 0 0 0 0
i=2: 34 32 30 29
i=3: 0 0 0 0
i=4: 0 0 3 9
i=5: 63 63 63 63
i=6: 59 59 59 54
i=7: 23 23 23 23
i=8: 54 55 56 57
i=9: 15 15 15 15
INFO: FOUND block structure at offset 8

Sonic found
blocksize will be 10 bytes

Synchronizing data WITHOUT Delimiter
TEST Pt 1
Pass 2
 (bytesavailable=0) --fail++ (result=1) . TEST Pt 2
INFO: Now trying undelimited mode (Test Pt 3)
bytesavailable=100
i=0: 44 40 35 31
i=1: 0 0 0 0
i=2: 44 40 36 30
i=3: 0 0 0 0
i=4: 21 23 23 23
i=5: 0 0 0 0
i=6: 53 7 61 61
i=7: 23 23 23 23
i=8: 51 52 53 54
i=9: 15 15 15 15
INFO: FOUND block structure at offset 8

Sonic found
Getting sonic status (CSAT3)...
...found prompt [>] in ?
 Received length of status (1)
 Status (Ss 0
?)
 Received length of status (1)
 Status (Ss 0
?)
 Received length of status (1)
 Status (Ss 0
?)
 Received length of status (1)
 Status (Ss 0
?)
 Received length of status (1)
 Status (Ss 0
```

```
?)
 Received length of status (1)
 Status (
s 0
?)
 Received length of status (1)
 Status (S0g00T1836U)
Now determining the Sonic Status
 Decoding Status: S0g00T1836U
 The trigger source is the CSAT3 Timer
 Execution Parameter g corresponds to 60 Hz -> 10 Hz
 Analog Range is OFF
 Data Status: Good Data
 CSI Mode
 CSAT3 serial no is 1836
 Unprompted Mode

Synchronizing data WITHOUT Delimiter
TEST Pt 1
Pass 3
 (bytesavailable=0) --fail++ (result=1) . TEST Pt 2
INFO: Now trying undelimited mode (Test Pt 3)
bytesavailable=100
i=0: 35 39 46 52
i=1: 63 63 63 63
i=2: 36 20 12 9
i=3: 0 0 0 0
i=4: 48 40 31 25
i=5: 62 62 62 62
i=6: 55 60 8 17
i=7: 22 22 22 22
i=8: 31 32 33 34
i=9: 15 15 15 15
INFO: FOUND block structure at offset 8

Sonic found
Host: Moxa

Synchronizing data WITHOUT Delimiter
TEST Pt 1
Pass 4
 (bytesavailable=0)  (result=1) . TEST Pt 2
INFO: Now trying undelimited mode (Test Pt 3)
bytesavailable=100
i=0: 32 50 63 30
i=1: 0 0 63 63
i=2: 39 44 50 49
i=3: 0 0 0 0
i=4: 40 54 63 63
i=5: 62 62 62 62
i=6: 34 40 58 14
i=7: 22 22 22 22
i=8: 27 28 29 30
i=9: 15 15 15 15
INFO: FOUND block structure at offset 8

Alarm signal received.
COMMENT     This is a sample sonicrc file by Werner Eugster
COMMENT     which is used for the CarboEuropeIP data aquisition systems
DATAPATH    ./
COMMENT     the old sonic anemometer has 5, the new HS sonic 6 analog channels
ANALOGS     0
COMMENT     MODE is ignored; leave it with a 1
```

```
MODE        1
COMMENT     the old sonic anemometer works with 4800, 9600 or 19200
COMMENT     the new HS sonic anemometer works with 9600, 19200 or 38400
BAUDRATE    4800
COMMENT     CHANGEFILE is an integer; after ... hours a new file will be opened
CHANGEFILE  6
COMMENT     FILECODE is one character that will appear in the filename ext.
FILECODE    b
LOGFILE     Moxa.log

COMMENT     below are additional settings for the new Solent HS sonic
COMMENT     AVERAGE 1 is 100 Hz, AVERAGE 5 is 20 Hz, AVERAGE 10 is 10 Hz
AVERAGE     5
COMMENT     define where IRGA data should be tanken from for averaging
COMMENT     valid options are: ANALOG, DIGITAL, BOTH or NONE
IRGA        DIGITAL
COMMENT     IRGA 2nd order conversion from millivolts to concentration
COMMENT     introduced in version 3.01; conc = a0 + a1*x + a2*x^2
COMMENT     format: three floating point numbers for a0, a1, and a2
COMMENT     only used for analog channel measurements
IRGA_CO2    0.0 1.0 0.0
IRGA_H2O    0.0 1.0 0.0
FILENAME=./2009072001.b42
Error sending message to queue with sendtoqueue()
Error sending message to queue with sendtoqueue()
Error sending message to queue with sendtoqueue()
correctly setting back the record numbering from 63 to 0.
```

Explanation: after the sonic was found, communication was established, the sonic anemometer is configures. Please note that it will use the sampling rate that you specify in `sonic.rc` (see EXEPARA in Section 8). If nothing is specified, then it will use mode **g** as you can see in the output above in the line

```
Execution Parameter g corresponds to 60 Hz -> 10 Hz
```

References

Spiegel, J. K., P. Zieger, N. Bukowiecki, E. Hammer, E. Weingartner, and W. Eugster (2012) Evaluating the capabilities and uncertainties of droplet measurements for the fog droplet spectrometer (FM-100). *Atmospheric Measurement Techniques* **5**, 2237–2260.