RBR

rbr-global.com

Logger2 Command Reference

Generation: Early 2015

F/W Type: 103

RBR0001963revB

Table of Contents

1 Introduction

This document applies *only* to RBR hardware that responds to the id (page 97) command with a **fwtype** parameter value of 103. When connected to the Ruskin software, this is identifiable via the 'Generation' label which reads "Early 2015".

1.1 Document version history

Ed.	Date	Rev	Notes
JmL	07Apr2017	B	New fast sampling periods added to sampling (page 35) command Added channels temp11, temp16, temp17, doxy23, doxy27, opt_07, opt_14 channels type.
JmL	06Deb2017	A	New numbering scheme for documentation.
JmL	01Feb2017	8.0	Documented Deployment Header (page 121) version 1.014. ddsampling (page 50) command added. Updated events for directional dependent sampling (EasyParse format events markers (page 116) and Standard format events markers (page 133)). Updated sampling (page 35) command. Updated error codes for verify (page 61), regimes (page 44) and regime (page 46) commands.
JmL /GmJ	22Nov2016	7.0	Documented Deployment Header (page 121) version 1.013. sensor (page 91) command documentation updated. settings (page 83) command documentation updated. Added acc_00, mag_00, echo01, alti00, doxy22 and doxy23 to list of supported channel types. Added distance calculation example. Added O2 concentration compensation and air saturation calculation example.

Ed.	Date	Rev	Notes
GmJ	26Jul2016	6.0	Documented the simulated data mode; see the simulation (page 67) command, id (page 97) command, and version 1.011 of the deployment header. Added doxy13, pres23, temp13, tran02, cond07, temp14, phas00, fluo35 channel types. Updated names reported by **channelslist** parameter to outputformat (page 52) command Minor improvement to description of channel (page 78) command. Corrected description of **Serial mode** in the Deployment Header (page 121) section.
JmL /GmJ	20Nov2015	5.0	Added peri00, peri01, bpr_08, bpr_09 channel types. Updated quick start documentation for High resolution BPR for cabled ocean observatories. Added example for deri_bprpres and deri_bprtemp. Updated documentation for **bpr** (page 86)command. Added new sensor (page 91) command. Added E0509 to list of error messages. Updated description of Deployment Header to version 1.010. Updated description of calibration (page 88) command for number of coefficients which may be changed. Added a short summary table of Deployment Header versions. Updated quickstart for float integration (page 18). Updated documentation for **regime** (page 46)command. Updated documentation for EasyParse format (page 113). Various formatting improvements.

Ed.	Date	Rev	Notes
JmL /GmJ	17Sep2015	4.0	Added **offsetfromutc** and **speccondtempco** parameters to settings (page 83) command and description of deployment header. Documented effect of the clock (page 31) command on **offsetfromutc**. Added **cond03**, **scon00** channel types. Added example for specific conductivity equation, **deri_speccond**. Updated and corrected description of **corr_cond** equation. Updated upper limit for sample period within bursts. Added description of factory default settings for serial (page 93) command. **caltext03** / **caltext04** output formats updated in the outputformat (page 52) command. Added **sos_00**, **bpr_04**, **bpr_05**, **bpr_06**, **bpr_07** channel types. Added speed of sound equation example.
JmL /GmJ	12Jan2015	3.0	Events marker corrected for threshold start/pause and switch to internal/external battery. Security updated for the **settings** command. Error conditions which may replace channel values in real time output are now documented under the outputformat (page 52) command. Added quick start for BPR users on ocean cabled observatories. **bpr** command updated.

Ed.	Date	Rev	Notes
GmJ	19Dec2014	2.0	Added the Section "Parsing logger responses". Changed 'off' to **tristate** for the **streamserial aux1_sleep** parameter. Added suggestions for failure recovery procedures to description of **status** command (also referenced from the **enable** command). Corrected description of **sleep** command. Added description of the **sleepafter** option to the **fetch** command. Description of equation for temperature compensation of pressure updated to describe cubic correction. Clarified the number format used for gains, described in the **channel** command. Updated or removed references to older channel types and equations for pressure and temperature. Section "Other deployment settings" renamed "Real time data". Description of **altitude** command moved to section "Other information". Updated description of command **regime**. Updated description of command **status**. Quickstart for float integration updated. Added channel type **cnt_00**. Added notes about channel type **cnt_00** to the sections "Integrating with a profiling float", "EasyParse format events markers" and "Standard format events markers". Added **measurement_count** to the **channelslist** decription in the **outputformat** command.

Ed.	Date	Rev	Notes
GmJ / JmL	24Nov2014	1.0	Modernized descriptions of **now**, **stop** commands. Documented the **reboot** command. Documented the new **sleep** command. Added descriptions of the AUX1 control parameters to the **streamserial** command Documented new error codes to **verify** and **enable** commands. Updated description of **memformat** and **enable** commands. Updated table of supported channel types. Updated tables of error codes. Updated description of deployment header, v1.008. Added descriptions of Rinko (B) equations. Removed all redundant references to old firmware and hardware revisions. Minor corrections to formatting, spelling, etc. Updated **outputformat** command to reflect new output formats available. Updated **regime** command with a binsize of 0. Updated description of standard format events markers.
GrJ	24Oct2014	n/a	Initial release (forked from 0000161rev14) of "Early 2014" document

1.2 Formatting

1. Examples of literal input to and output from the logger are shown in **bold type**.
2. In examples of dialogue between the logger and a host, input to the logger is preceded by **>>**, while output from the logger is preceded by **<<**. These characters must not actually be included in commands or expected in responses.
3. Some examples of command dialogues contain descriptive comments which are not part of the command or response. These start with a percent character, %.
4. When an item or group of items is optional, it is enclosed in [square brackets].
5. Where an item can be only one of several options, options are separated by vertical **|** bars.
6. Place holders for variable fields are in <*italics enclosed in angle brackets*>

7. Lists are used for unknown or variable numbers of items, or to abbreviate large numbers of options, and are specified by giving a first example of an item, followed by a comma and ellipsis, such as <*example-value*>, …

1.3 Security

There are several levels of access control to logger commands, although "levels" is perhaps not the best word, as more than one can apply:

1. **Open** commands can be executed without restriction.
2. **Unsafe** commands are those which the logger will not execute if logging is in progress. For example, the sampling period cannot be changed in the middle of a deployment.
3. **Protected** commands are those which might be considered 'dangerous'; for example, clearing the memory. These have a safety guard on them; see the permit (page 105) command.

1.4 Command Processing and Timeouts

Commands may be sent to the logger via either the USB-CDC port or a true serial port (RS-232 or RS-485). With a few exceptions and minor differences, both ports are intended to offer the same functionality, but can not be used for command input simultaneously. If this is attempted, then either one of the ports will not respond, or there will be a 'busy' message:

E0101 command parser busy

1.4.1 Command Entry

A *potential* command is considered to begin when its first character is received. For the serial port this is straightforward; for the USB it is hard or impossible for the CPU to 'see' how the messages are packaged, but the overall effect is similar. In both cases the potential command has been received once the logger sees a termination character; either one of <**CR**> (0x0D) or <**LF**> (0x0A). Combinations of the two characters are dealt with as follows:

1. <CR><LF>
2. <LF><CR>
3. <CR><CR>

4. <LF><LF>

For (1) and (2), the second character is considered redundant and is discarded. For (3) and (4), the second character in either case is treated as an empty command, which simply provokes the logger's prompt, **Ready:** (see the prompt (page 106) command).

The received message may or may not form a valid command; errors detectable by the logger will vary from one command to another, but some of the common, general errors include:

1. E0102 invalid command '<unknown_text>'
2. E0107 expected argument missing
3. E0108 invalid argument to command: '<unknown_text>'

See Error messages (page 185) for a complete list.

1.4.2 Timeouts, Output Blanking and Power Saving

Wakeup

All RBR instruments sleep as much as possible. Interaction requires that the instrument be woken up first, then a series of commands issued. After a 10-second idle timer elapses, the instrument will return to the low-power sleep mode.

The wakeup procedure is to send a single character; carriage return <**CR**> (0x0D) is the recommended choice. Over the USB link, the response is usually immediate. Over the Serial link, this first character may not be completely received by the instrument due to the non-zero wakeup time required, and it may be seen as a garbage character. However, the instrument itself ignores all garbage characters received immediately after wakeup, and so will not return any errors.

After the initial <**CR**> character, a 10ms pause should be used. Following this, the instrument is fully ready to receive any valid command.

In the RBR Ruskin software which is used by end-customers, the following is an example of the wakeup sequence used:

```
>> <CR>              % nothing may be returned by this character, but the logger
will start to wakeup
[10ms pause]          % the logger completes the wakeup procedure
>> id<CR>            % the id command is a useful initial command as it replies
with confirmation of the instrument connection
<< id model = RBRduo, version = 1.000, serial = 050050, fwtype = 101     % this
is the reply from the instrument
<< Ready:             % this is the Ready prompt, which may or may not be
included, depending on the state of the 'prompt' command
```

Output blanking

When the first character of a potential command is received, a 10-second timeout is started. This timeout serves two purposes: output blanking and power saving.

As soon as the logger knows it may be about to receive a command, any output which it could autonomously generate (such as streamed sample data) is suppressed. This is to avoid confusing the host, which has just sent a command and may be expecting a particular form of response. Until the logger has processed the command and sent the response, any other outputs will be suppressed. Output such as streamed data may appear *in between* received commands, but not while a command is being received or processed.

This 'output-blanking' state does not persist forever; if the 10-second timeout expires before a command terminator is seen, outputs such as streamed data are permitted again. This poses no problem for machine generated commands, but can be limiting for commands typed manually at a terminal.

The output blanking behaviour does *not* apply to the very first (potential) command received after the logger is woken from a quiescent state. For this command, outputs such as streamed data may continue to appear while the command is being received. This exception prevents, for example, random noise input from suppressing required data output. The logger will not invoke the output blanking behaviour until it has seen at least one valid command, at which point it can reasonably assume that a valid host is genuinely trying to communicate with it. Empty commands (isolated <**CR**> and/or <**LF**>) do not count as "valid commands" for this purpose.

Power saving

The second purpose of the 10-second timeout is to minimize power consumption. If no valid, terminated command is received within the timeout, the communication returns to a quiescent state. This means that it discards any incomplete input, restarts the "valid command" timeout, and will start afresh with the next input character.

In the case of the serial port, it also allows the transmit hardware to be turned off to save power; indeed, if the logger has no other tasks to perform the entire instrument will enter a low power sleep mode.

The USB port is different in this respect, because the logger can draw enough power from the connection to run most of its basic functions. As long as the USB is connected the logger remains 'awake' and responsive to commands; no hardware is shut down. However, expiry of the 10-second timeout still resets the command processor's behaviour with respect to its 'memory' of valid commands, incomplete input and output blanking.

Changing the Timeout

The timeout referred to in the above sections has a default value of 10 seconds in all instruments. As noted, this is short enough to make manual typing of commands difficult in some cases. For instruments with firmware version 1.130 or later, the timeout can be changed; refer to the description of the **inputtimeout** parameter for the settings (page 83) command for more details.

1.4.3 Parsing logger responses

To implement robust automated parsing of responses to logger commands, there are some important points to consider.

- Do not assume the upper-case/lower-case nature of the responses will match those in the command. For example,

  ```
  >> MEMINFO USED
  << MEMINFO used = 0
  ```

  It is good practice to make parsing insensitive to the case of the responses.

- Do not assume that parameters will be reported in the same order they were requested. For example,

  ```
  >> meminfo size used remaining
  << meminfo used = 0, remaining = 132120576, size = 132120576
  ```

  It is good practice to check each <*key*> = <*value*> pair for the <*key*> of interest until all searches are satisfied.

- Be aware that future versions of the instrument firmware may report parameters that are not documented here, and that the reporting order may change. For example,

  ```
  >> meminfo
  << meminfo used = 0, remaining = 132120576, size = 132120576
  ```

  is the current behaviour, but a future version might respond as follows:

  ```
  
  ```

```
>> meminfo
<< meminfo used = 0, remaining = 132120576, capacity = 132120576, size =
132120576
```

Again, it is good practice to check all <*key*> = <*value*> pairs, and be prepared to ignore <*key*>s which are not recognized.

- Do not assume that numeric fields will always have the same number of digits. Even parameters whose values might be expected to remain fixed can change if the logger is used in a different configuration. For example, an instrument with an auto-ranging channel might behave as follows:

```
>> channel 4 gain
<< channel 4 gain = auto

>> channels readtime
<< channels readtime = 1890

>> channel 4 gain = 20

<< channel 4 gain = 20.0

>> channels readtime

<< channels readtime = 350
```

This is true even when parsing data values with a well specified format. For example, even though reporting of values may be specified to contain four decimal places (eg. 21.7325), parsing this number without assuming anything about how many digits there are is more robust.

- When parsing numbers of any sort, use the most inclusive format which is practical. In principle, parsing everything as a double precision floating point number would almost always work (one exception being the 64-bit integers used for timestamps in EasyParse format: see the paragraph "Sample timing" in Section "Sample data EasyParse format (page 113)"). Recognizing that such an approach is overkill and may add unacceptable overhead in some applications, parsing all integers as signed 32-bit quantities and all floating point values as single precision (IEEE-754 32-bit) numbers would be satisfactory. It may be assumed that numbers are integers unless the documentation or examples make it clear that they are floating point values.

  Note that some commands accept and respond with date/times which look like very large integers, but which have an implicit special format. For example, 20150401120000 represents noon on 1st April 2015. These cases are usually clear from the context.

2 Quick start

This section details commands sent to (**>>**) and responses received from (**<<**) the logger in order to perform a desired action. These do not cover an exhaustive list of possibilities but are intended to be a starting point from which to start interacting with the logger.

The **prompt** state has been disabled (**prompt state = off**) for all of these examples. If it was enabled you should expect a **Ready:** prompt following all of the logger's responses.

2.1 Enabling continuous sampling

Start sampling for 24 hours at 1 sample per second starting from the current time of December 1, 2012 12:00 am.

```
>> now = 20121201000000
<< now = 20121201000000
>> starttime = 20121201000000
<< starttime = 20121201000000
>> endtime = 20121202000000
<< endtime = 20121202000000
>> sampling mode = continuous, period = 1000
<< sampling mode = continuous, period = 1000
>> verify
<< E0402 memory not empty, erase first, verify = stopped
>> enable erasememory = true
<< enable = logging
>> status
<< status = logging
```

2.2 Enabling wave sampling

Collect 1024 samples at a rate of 4Hz, every 30 minutes starting from December 1, 2012 12:00 am. The start time is set for 24 hours in the future of the current logger time. The end time is set for December 31, 2012 12:00am.

```
>> now
<< now = 20121130000000
>> starttime = 20121201000000
<< starttime = 20121201000000
>> endtime = 20121231000000
<< endtime = 20121231000000
>> sampling mode = wave, period = 250, burstlength = 1024, burstinterval = 1800000
<< sampling mode = wave, period = 250, burstlength = 1024, burstinterval = 1800000
>> altitude = 0.26
<< altitude = 0.26
>> verify
<< E0402 memory not empty, erase first, verify = stopped
>> enable erasememory = true
<< enable = pending
>> status
<< status = pending
```

2.3 Enable serial streaming from serial port

The serial streaming feature may not be available on all instruments. If the feature is not supported you will receive the '**E0109 feature not available**' message in response to the streamserial (page 57) command. Here we set the baud rate to 115200 and the output format to caltext01 (which is a calibrated output) before enabling the streaming state.

```
>> serial
<< serial baudrate = 19200
```

```
>> serial baudrate = 115200
<< serial baudrate = 115200
% This response is sent at the old baudrate, 19200Bd.
% The host must now change its baudrate to 115200Bd.
>> outputformat support
<< outputformat support = caltext01, caltext02, caltext03, caltext04
>> outputformat type = caltext01
<< outputformat type = caltext01
>> streamserial state = on
<< streamserial state = on
```

2.4 Download stored data

In order to download data we must know how much data is stored by using the **meminfo used** command. Then we can download either the entire amount in one transfer or in multiple chunks of data. Below is an example where we download in multiple smaller chunks. Each chunk of downloaded data is followed by a two byte CRC error check which must not be included when parsing the data. For parsing the downloaded data see the *Format of Stored Data (page 112)* section.

```
>> meminfo used
<< meminfo used = 2000
>> read data 1 500 0
<< data 1 500 0<cr><lf><bytes[0…499]-of-data><crc>
>> read data 1 500 500
<< data 1 500 500<cr><lf><bytes[500…999]-of-data><crc>
>> read data 1 500 1000
<< data 1 500 1000<cr><lf><bytes[1000…1499]-of-data><crc>
>> read data 1 500 1500
<< data 1 500 1500<cr><lf><bytes[1500…1999]-of-data><crc>
```

2.5 Downloading an EasyParse dataset

EasyParse format (calbin00) datasets are intended to make the process of parsing and subsequent viewing or analysis of data extremely simple. In the Standard memory format (rawbin00), calibration coefficients, deployment settings, raw ADC data, and logger events are all combined in a single binary block, which requires substantial amounts of code to untangle. The EasyParse format contains two separate blocks, one of which is for data alone, and the other containing logger events. Many applications only require the data block, further simplifying the process.

There are costs associated with the EasyParse format: memory consumption, and loss of certain post-processing capabilities (mainly post-deployment calibration). These costs are often not significant obstacles.

2.5.1 Starting a deployment using EasyParse

Before the logger is enabled, it must be instructed to use the EasyParse format using the memformat (page 75) command, by specifying the **newtype** parameter. This step does not need to be repeated before every deployment unless the format is to be changed. The format can not be changed while logging is in progress.

```
>> memformat newtype = calbin00
```

This can be verified after the logger has been enabled, again using the memformat (page 75) command:

```
>> memformat
<< memformat type = calbin00
```

The readings made by the logger are stored in dataset-1, as a series of records containing a timestamp, and readings for each channel. The readings are IEEE single precision floats, representing engineering unit data (degC, mS/cm, dbar, etc). Often, this is the only dataset required for a download. However, all logger events are stored in dataset-0, and these may be of interest in certain cases. In particular, the cast detection events and WiFi module enable events can be useful markers that permit a smart host to download just a small section of the data (perhaps 'the latest profile' or 'all data since the last time the WiFi module was enabled').

Downloading the data block in dataset-1 uses the read (page 72) command. First find out how much data is in use:

```
>> meminfo dataset 1 used

<< meminfo dataset = 1, used = 2000
```

Read that data in a single 2000 byte chunk:

```
>> read data 1 2000 0
<< data 1 2000 0<cr><lf><bytes[0…1999]-of-data><crc>
```

In addition to the data block, there is another essential piece of information - the currently active channel list. The length of this list, and the ordering of the channels in it, will be an essential part of parsing the data block (remember the timestamp-value1-value2-value3...-valueN record structure).

Request the active channel list:

```
>> outputformat channelslist
<< outputformat channelslist = temperature (C), pressure (dbar)
```

Now we know there are just two channel readings in the data, so each sample is composed of a timestamp (8 bytes, unsigned 64-bit integer, milliseconds since the Unix epoch at 1970-Jan-01 00:00:00) followed by one reading each of temperature and pressure (4 bytes each, IEEE floats).

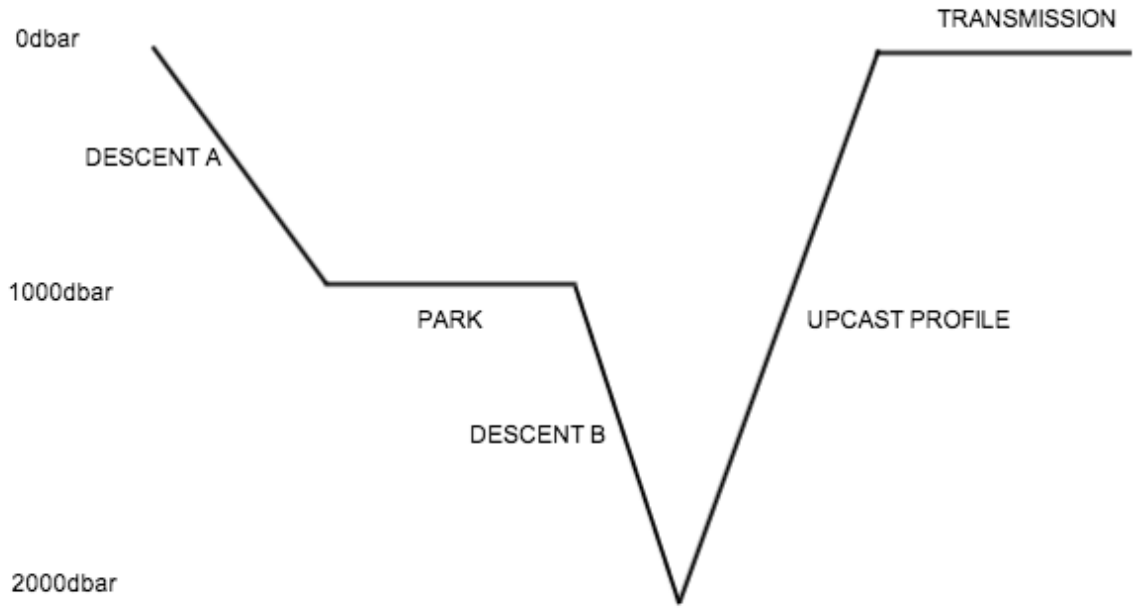
Converting the data at this point should be straightforward.

2.6 Integrating with a profiling float

2.6.1 Introduction

There are a number of dedicated features in the Logger2 products aimed at profiling floats. The primary requirement of these vehicles is to have multiple sampling regimes that are enabled according to depth.

The typical Argo profiler might be set up with the following behaviour:

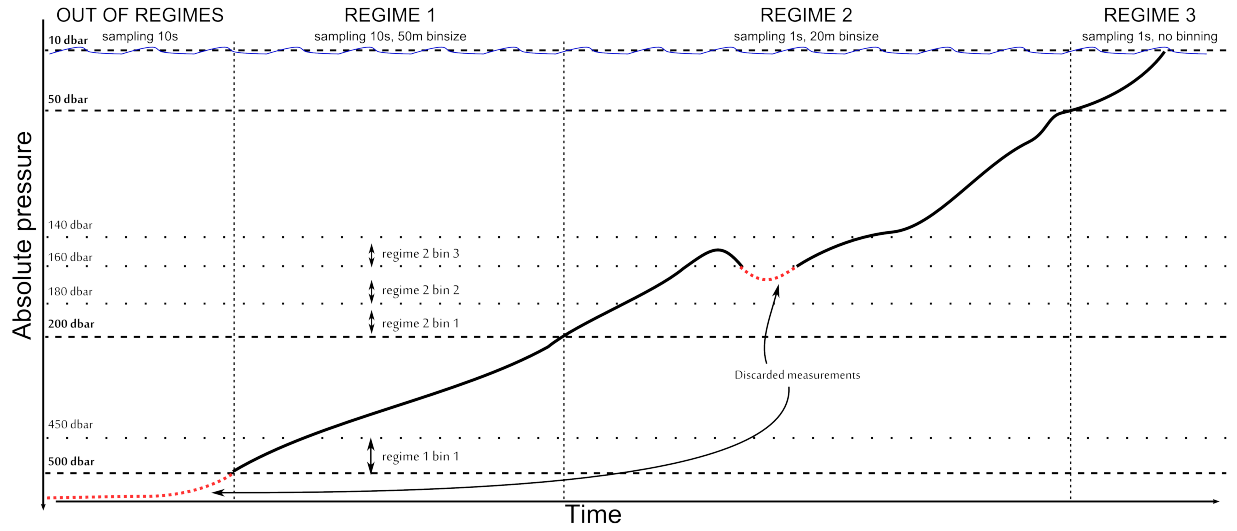


2.6.2 Control of buoyancy

For the majority of this 10 day schedule, the RBR instrument is used purely as a depth sensor, providing input to the buoyancy engine and float controller. This is typically done interactively using the fetch (page 104) command, which can be performed at any time, regardless of whether the RBR instrument logging schedule is enabled or not. The RBR will automatically fall asleep after an idle timeout of a few seconds, but in order to minimize the power consumption, it is recommended that the command **fetch sleepafter = true** is used. This will override the idle timeouts and does not affect any ongoing deployment. In order to wake up the instrument, a <cr> character should precede any command. This is a useful practice in any situation when it is not clear whether or not the instrument is asleep.

2.6.3 Setup for deployment

The real science of the Argo profiler occurs during the upcast, typically from 2000dbar to the surface. For historical and scientific reasons, this is often a multi-stage ascent, where the sampling and binning requirements change according to depth. The expanded figure below shows an example of a typical ascent setup.



As is clear, there are three distinct sampling regimes that are required. Each of them has a boundary, a sampling speed, and an averaging bin size.

- The boundary determines when the regime comes into effect (dbar).
- The sampling speed dictates the measurement rate that is used internally (msec).
- The bin size dictates the amount of water column (in dbar) over which the samples will be averaged and stored.

To accomplish this task, a combination of the regimes (page 44) and three regime (page 46) commands are needed.

First, one should tell the logger that it will be ascending (decreasing water pressure) using three sampling regimes, and the absolute pressure serves as reference.

```
>> regimes direction = ascending, count = 3, reference = absolute
<< regimes direction = ascending, count = 3, reference = absolute
```

Configure the regime closest to the seabed (10s sampling rate, 50m bin, bottom boundary 500 dbar):

```
>> regime 1 boundary = 500, binsize = 50, samplingperiod = 10000
<< regime 1 boundary = 500, binsize = 50, samplingperiod = 10000
```

Then the middle one (1s sampling rate, 20m bin, bottom boundary 200 dbar):

```
>> regime 2 boundary = 200, binsize = 20, samplingperiod = 1000
<< regime 2 boundary = 200, binsize = 20, samplingperiod = 1000
```

Then the regime closest to the sea surface (1s sampling rate, no binning, bottom boundary 50 dbar):

```
>> regime 2 boundary = 50, binsize = 0, samplingperiod = 1000
<< regime 2 boundary = 50, binsize = 0, samplingperiod = 1000
```

Finally put the logger in regimes mode:

```
>> sampling mode=regimes
<< sampling mode=regimes
```

With this setup, the logger will only start recording data once the boundary of the first regime is crossed. In this example, if the RBR instrument is enabled while the float is at 700 dbar, then starts to ascend, no data will be stored until 500 dbar. If the float is at 490 dbar when the RBR instrument is enabled, sampling will start immediately as the first regime is already in effect.

Instruments with firmware version 1.100 or later may also support a derived data channel, type **cnt_00**, which contains a count of the number of measurements in the averaged bin when in the regimes sampling mode. As with any other channel, it may be turned on or off as needed; see the channel command. The benefit of turning it on when storing data in EasyParse format is that the value is then available when the main sample data in dataset-1 is downloaded. Otherwise, the dataset containing the events must also be retrieved if these values are needed.

The **cnt_00** channel is most useful in **regimes** mode; if it is turned on for other sampling modes, it will report as follows:

- in **average** or **tide** modes, a count of the number of measurements contributing to the average; this will usually be the same as the **burstlength**.
- in **continuous**, **burst** or **wave** modes, the value 1, as each sample stored has only one measurement associated with it.

See the sampling (page 35) command for further information about sampling modes.

Unlike other CTDs, RBR instruments can sample through surface waters without concerns. In fact, making measurements in air can provide a reference drift measurement for conductivity, and a potentially useful barometric pressure reading as well.

> When switching between regimes with different sampling rates, the logger may not sample for up to 2 seconds.

Ensure the logger will use the EasyParse format (page 113) (calbin00).

```
>> memformat newtype = calbin00
<< memformat newtype = calbin00
```

Starts the logger while ensuring the memory is cleared first.

```
>> enable erasememory = true
<< enable = logging
```

There will also be a bin in progress which never completes, once the float has risen to the surface. However, issuing the stop (page 66) command will flush the final bin to memory, regardless of whether it is complete or not.

2.6.4 Downloading data after deployment

Downloading the data from the instrument can be done while a schedule is still enabled, but this requires careful housekeeping to keep track of what data has been added to the dataset since the last retrieval. In this example, the logger is stopped before the download commences.

Stop the current deployment

```
>> stop
<< stop = stopped
```

Determine how much memory has been used

```
>> meminfo used
<< meminfo used = <bytes-used-in-dataset-1>
```

Now loop over the data to download it in chunks

```
>> read data 1 <chunk-size> 0
<< data 1 <chunk-size> 0<cr><lf><chunk-size data bytes><crc>
>> read data 1 <chunk-size>
<< data 1 <chunk-size> <1*chunk-size><cr><lf><chunk-size data bytes><crc>
>> read data 1 <chunk-size>
```

```
<< data 1 <chunk-size> <2*chunk-size><cr><lf><chunk-size data bytes><crc>
...
...
>> read data 1 <chunk-size>
<< data 1 <final-chunk-size> <(n-1)*chunk-size><cr><lf><chunk-size data bytes><crc>
```

The data returned by the read command has a CRC value at the end. This can be used to verify the integrity of the download, but should **not** be stored. All chunks should be concatenated together.

Parsing the resultant data block can be done according to the description in the "EasyParse format (page 113)" section Briefly, each record consists of an 8-byte (64-bit) unsigned integer timestamp, and a 4-byte IEEE-754 floating point value for each channel.

2.6.5 More details on calculation

Please note that samples taken in a bin whose upper limit as already been crossed, are discarded as it is shown on the previous example figure.

2.7 High resolution BPR and frequency counters for cabled ocean observatories

The following applies to BPR loggers manufactured after November 2015 and all loggers having frequency counter channels (**peri_XX**). For the former implementation please refer to BPR in High Resolution mode for Cabled Ocean Observatories (page 26).

2.7.1 Introduction

Logger with frequency counters channels can achieve the very high resolution (10 ppb). In order to maximize the resolution some steps are required. This chapter present the main points to follow in order to ensure the best performance for loggers using frequencies counters (and/or BPR channels). When sampling at rate 1Hz or slower, the frequency counter board will integrate the signal over around 800 ms, with faster sampling speeds, the whole period of time between samples is used as integration time.

2.7.2 Deploying a Frequency counter/BPR logger running continuously at 1Hz and streaming over serial

> ⚠ To avoid any loss of resolution, calibrated output formats **caltext03** and **caltext04** should be used (see **outputformat** command). Indeed, the other calibrated output formats outputs 6 significant digits which is not enough to cover the full resolution achievable.

We should then first make sure the output format is in caltext04 (or caltext03) and we can enable streaming while logging:

```
>> outputformat type = caltext04
<< outputformat type = caltext04
>> streamserial state = on
<< streamserial state = on
```

We can then setup the deployment in continuous mode at 1Hz, the starttime and the endtime (here between the 01/12/2012 and the 02/12/2012):

```
>> sampling mode = continuous, period = 1000, gate = none
<< sampling mode = continuous, period = 1000, gate = none
>> starttime = 20121201000000
<< starttime = 20121201000000
>> endtime = 20121202000000
<< endtime = 20121202000000
```

> ⚠ In High Resolution EasyParse stored data format can NOT be used. Indeed, the EasyParse format stores data as single (32-bits) floats, hence reducing the achievable resolution.

We can now enable the datalogger:

```
>> enable erasememory = true
<< enable = logging
>> status
<< status = logging
```

The datalogger now outputs continuously the measurements:

```
<< 2015-11-23 18:37:48.000, 30.39588279090822e+006, 5.825177871156484e+006,
17.1028520e+000, 22.4525380e+00
```

2.7.3 BPR channels

Ordered High Resolution RBR BPR are configured to provide 2 frequency counters channels and 2 BPR channels per BPR sensor : pressure signal period, temperature signal period, calculated pressure, calculated temperature. The datalogger stores only the frequency measurements (BPR channels being derived).

The channel command gives back:

```
>> channel 1
<< channel 1 type = peri00, module = 96, status = on, latency = 900, readtime = 858,
equation = lin, userunits = ps
>> channel 2
<< channel 2 type = peri01, module = 97, status = on, latency = 900, readtime = 858,
equation = lin, userunits = ps
>> channel 3
<< channel 3 type = bpr_08, module = 243, status = on, latency = 0, readtime = 0, equation
= lin, userunits = dbar
>> channel 4
<< channel 4 type = bpr_09, module = 244, status = on, latency = 0, readtime = 0, equation
= lin, userunits = C
```

And the calibration command returns:

```
>> calibration 1
<< calibration 1 type = peri00, datetime = 20000401000000, c0 = 20.000000e+006, c1 =
10.000000e+006
>> calibration 2
<< calibration 2 type = peri01, datetime = 20000401000000, c0 = 5.0000000e+006, c1 =
2.5000000e+006
>> calibration 3
<< calibration 3 type = bpr_08, datetime = 20151123120721, x0 = 5.8310300e+000, x1 =
-24.514030e+003, x2 = -573.64115e+000, x3 = 76.129280e+003, x4 = 35.688000e-003,
x5 = 0.0000000e+000, x6 = 30.413170e+000, x7 = 664.14899e-003, x8 = 58.803408
e+000, x9 = 180.91160e+000, x10 = 0.0000000e+000, n0 = 1, n1 = 2
>> calibration 4
<< calibration 4 type = bpr_09, datetime = 20151123120722, x0 = 5.8310300e+000, x1 =
-3.8981210e+003, x2 = -10.493120e+003, x3 = 0.0000000e+000, n0 = 2
```

If the Paroscientific Inc. transducer attached to the RBR BPR logger is changed, please refer to Example 12: deri_bprpres and deri_bprtemp channels (page 175) in order to change the calibration.

2.8 High Resolution BPR for cabled ocean observatories (older models)

⚠ The following applies to BPR loggers manufactured before November 2015 (except special orders for backward compatibility). Those previous loggers have BPR channels of types **bpr_00** to **bpr_07**.

2.8.1 Deploying a BPR logger running continuously at 1Hz and streaming over serial

First we have to check the current configuration of the BPR board in the datalogger.

```
>> bpr 1
<< bpr 1 serial = 130026, fullscale = 10000.0, u0 = 5.8618360e+000, y1 = -3.9391280
e+003,
y2 = -10.166080e+003, y3 = 0.0000000e+000, c1 = -43.615040e+003, c2 = -396.53600
e+000,
c3 = 132.49310e+003, d1 = 33.126998e-003, d2 = 0.0000000e+000, t1 = 29.869482e+000,
t2 = 1.2553540e+000, t3 = 62.896060e+000, t4 = 196.86850e+000, t5 = 0.0000000e+000,
paroscaldate = 20140731, integrationtime = 400, sensorsettling = 200, oversampling = 8
```

In this case the integration time is 400 ms where it could be increased up to 810 ms at 1Hz, the oversampling is set at 8, it can be increased up to 128 (reduces the internal jitter):

```
>> bpr 1 integrationtime = 810, oversampling = 128
<< bpr 1 integrationtime = 400, oversampling = 128
```

⚠ To avoid any loss of resolution, calibrated output formats **caltext03** and **caltext04** should be used (see **outputformat** command). Indeed, the other calibrated output formatsoutputs 6 significant digits which is not enough to cover the full resolution achievable.

Then we should make sure the output format is in caltext04 and we can enable streaming while logging:

```
>> outputformat type = caltext04
<< outputformat type = caltext04
>> streamserial state = on
<< streamserial state = on
```

In order to run at 1Hz, we should let the power to the sensor on all the time:

```
>> settings sensorpoweralwayson = on
<< settings sensorpoweralwayson = on
```

We can then setup the deployment in continuous mode at 1Hz, the starttime and the endtime (here between the 01/12/2012 and the 02/12/2012):

```
>> sampling mode = continuous, period = 1000, gate = none
<< sampling mode = continuous, period = 1000, gate = none
>> starttime = 20121201000000
<< starttime = 20121201000000
>> endtime = 20121202000000
<< endtime = 20121202000000
```

⚠ In High Resolution EasyParse stored data format can NOT be used. Indeed, the EasyParse format stores data as single (32-bits) floats, hence reducing the achievable resolution.

We can now enable the datalogger:

```
>> enable erasememory = true
<< enable = logging
>> status
<< status = logging
```

The datalogger now outputs continuously the measurements:

```
<< 2012-12-01 19:17:46.000 25.82445371896028e+006, 5.824454040266573e+006,
10.9584250e+000 , 24.3679000e+000
```

2.8.2 Processing the measurements

Ordered High Resolution RBR BPR are configured to provide four channels per BPR sensor : pressure signal period, temperature signal period, calculated pressure, calculated temperature. It is wise to save all the measurements from all the channels.

In the following examples, we will use the same logger as the one used for the previous example (same sensor).

The channel command gives back:

```
>> channel 1
<< channel 1 type = bpr_04, module = 96, status = on, latency = 210, readtime = 858,
equation = lin, userunits = ps
>> channel 2
<< channel 2 type = bpr_05, module = 97, status = on, latency = 210, readtime = 858,
equation = lin, userunits = ps
>> channel 3
<< channel 3 type = bpr_06, module = 98, status = on, latency = 210, readtime = 858,
equation = lin, userunits = dbar
>> channel 4
<< channel 4 type = bpr_07, module = 99, status = on, latency = 210, readtime = 858,
equation = lin, userunits = C
```

And the calibration command returns:

```
<< calibration 1 type = bpr_04, datetime = 20140829175524, c0 = 20.000000e+006, c1 =
10.000000e+006
>> calibration 2
<< calibration 2 type = bpr_05, datetime = 20140829175524, c0 = 5.0000000e+006, c1 =
2.5000000e+006
>> calibration 3
<< calibration 3 type = bpr_06, datetime = 20140829175525, c0 = 0.0000000e+000, c1 =
4.8440890e+003
>> calibration 4
<< calibration 4 type = bpr_07, datetime = 20140829175525, c0 = -2.0000000e+000, c1=
42.000000e+000
```

We will refer to those coefficients by c0_x and c1_x, x being the channel indice.

> ⚠ Those coefficients are not the Paros sensor calibration. They are calculated automatically by the logger when setting up the Paros sensor parameters via the **bpr** command.

In the following examples we will try to process the following line:

```
<< 2012-12-01 19:17:46.000 1058557540 367455000 2429074 674098176
```

Processing the measurements from pressure signal period and temperature signal period channels

The first step is to obtain the temperature period and the pressure period.

```
pressure period (picoseconds) = c0_1 + c1_1 * ( 1058557540 /(2^30) )
```

```
temperature period (picoseconds) = c0_2 + c1_2 * (367455000 /(2^30) )
```

To get to the decibars and degrees world (in pseudo code matlab)

X = temperature period in microseconds
T = pressure period in microseconds

U = X - U0;
Temperature = Y1.*U+Y2.*(U.^2)+Y3.*(U.^3); % temperature in degree Celsius
C = C1 +C2.*U +C3.*(U.^2);
D = D1+ D2.*U;
T0 = T1 + T2.*U + T3.*(U.^2) + T4.*(U.^3) + T5.*(U.^4);
T0sqoverTsq= (1 - (T0.*T0./(T.*T)));
Pressure_psia = C .* T0sqoverTsq .* (1 - D .* T0sqoverTsq); % pressure in psia
Pressure_dbar = (0.689475728*Pressure_psia); % pressure in dbar

In our example it gives 10.958425 dbar and 24.3679 °C.

Processing the measurements from pressure and temperature channels

```
pressure (dbar) = c0_3 + c1_3 * ( 2429074 /(2^30) )
```

```
temperature (°C) = c0_4 + c1_4 * (674098176 /(2^30) )
```

In our example it gives 10.958547dbar and 24.3677 °C.

This slight difference compared to the previous result is explained by the fact that the datalogger store the coefficients as 32 bits float (instead of 64 bits in MATLAB) even if 64-bit floating point numbers calculations are performed.

3 Commands

This chapter describes all of the logger commands.

3.1 Time and Schedule

3.1.1 now

Usage

>> clock [= <YYYYMMDDhhmmss>]

Security

Open, Unsafe (modifications not permitted while logging).

Description

Retrieve or set the logger's current date and time.

For firmware versions 1.210 or later, be aware that setting the date and time will erase the logger's record of the time zone. If this setting is required, it must be restored separately after updating the date/time. Simply reading the date/time does not affect the time zone setting. For more details, refer to the **offsetfromutc** parameter described under the settings (page 83) command.

For **fwtype = 103** only (see the id (page 97) command), the alternate name **clock** may be used instead of **now**, but for compatibility with all current instruments from RBR Ltd. the use of **now** is recommended.

Examples

```
>> now
<< now = 20110401120000
```

```
>> now = 20110401120130
<< now = 20110401120130
```

Errors

Error E0105

Date/time may not be modified while logging is in progress; reading is permitted.

Error E0108

The supplied argument was not a valid date/time.

3.1.2 starttime

Usage

>> starttime [=<YYYYMMDDhhmmss>]

Security

Open, Unsafe (modifications not permitted while logging).

Description

Retrieve or set the start date and time of the next deployment.

Examples

```
>> starttime
<< starttime = 20100314000000
```

```
>> starttime = 20110317120000
<< starttime = 20110317120000
```

Errors

Error E0105
Start time may not be modified while logging is in progress; reading is permitted.

Error E0108
The supplied argument was not a valid date/time.

3.1.3 endtime

Usage

>> endtime [= <YYYYMMDDhhmmss>]

Security

Open, Unsafe (modifications not permitted while logging).

Description

Retrieve or set the end date and time of the next deployment.

Examples

```
>> endtime
<< endtime = 20101101073000
```

```
>> endtime = 20120101000000
<< endtime = 20120101000000
```

Errors

Error E0105
End time may not be modified while logging is in progress; reading is permitted.

Error E0108
The supplied argument was not a valid date/time.

3.1.4 sampling

Usage

>> sampling [<parameter> =<value>,...]

Security

Open, Unsafe (modifications not permitted while logging).

Description

Allows various parameters to be reported or set for a sampling schedule. The <*parameter*>s currently supported are:

1. **schedule** [=<*index*>] reports or sets the index of the schedule to be read or modified. At the present time the logger supports only a single schedule, and the value of the <*index*> must always be 1.
2. **mode** [= continuous | burst | wave | average | tide | regimes] reports or sets the sampling mode for the current schedule.
    a. **continuous** mode is supported by all loggers; measurements are taken and stored at the specified period between the start and end times.
    b. **burst** mode is available for loggers which are so configured in the factory; measurements are taken and stored in bursts between the start and end times. The time between the start of two consecutive bursts is given by the **burstinterval**, the number of measurements in the burst is given by the **burstlength**, and the time between measurements within the burst is given by the **period**.
    c. **wave** mode is available for loggers which are so configured in the factory, and with one exception is identical to **burst** mode as far as the logger's behaviour is concerned. The two modes are distinct to allow host software to process the burst measurements for wave applications. The exception is that the **altitude** command is available *only* when the logger is capable of the **wave** sampling mode.
    d. **average** mode is available for loggers which are so configured in the factory. It works identically to **burst** mode, except that instead of storing every measurement in the burst, the average value of all measurements is computed and stored as a single sample value at the end of the burst.
    e. **tide** mode is available for loggers which are so configured in the factory, and is identical to **average** mode as far as the logger's behaviour is concerned. The two modes are distinct to allow host software to process the data for tide monitoring applications.

f. **regimes** mode is available for all loggers, but is intended for vehicle integration. Instead of a deployment containing a single sampling period, averaging burstlength, etc, the **regimes** mode permits multiple sampling periods, averaging bins, and sampling periods to be set for different environmental conditions-pressure regimes.

g. **ddsampling** mode is available for all loggers, but is mainly intended for vehicle integration. Instead of a deployment containing a single sampling period, the **ddsampling** mode permits two sampling periods depending on the the current direction of the logger (descending or ascending). See command ddsampling (page 50).

3. **period** [=<*period*>] reports or sets the time between measurements.
This is straight forward in **continuous** mode; in any of the other modes it is the time between continuous measurements *within* the burst. The <*period*> is specified in milliseconds.

   Values for 1Hz sampling or slower are supported by all loggers, and must be given in multiples of 1000. In **continuous** mode the permitted range is typically 1000 (one second) to 86400000 (24 hours), in increments of 1000. Some loggers may have a lower limit which is more than 1000 if an attached sensor has a very slow data acquisition time. For all other modes the upper limit is 255000 (about 4 minutes); this is unlikely to be a constraint in practice, as the period is the time between measurements *within* the burst.

   If the logger is configured to support fast (sub-second) measurement periods for the selected mode, the <*period*> must correspond to an exact frequency in Hz, to the nearest millisecond. Permitted values may be further constrained as follows, according to the firmware version:

   Firmware versions up to and including 1.360:
   |fast 6 - **500**(2Hz), **333**(3Hz), **250**(4Hz), **200**(5Hz), and **167**(6Hz)
   |fast12 - All |fast6 values plus **83**(12Hz).
   Firmware versions later than 1.360:
   |fast 6 - **500**(2Hz), **250**(4Hz), and **167**(6Hz)
   |fast16 - All |fast6 values plus **125**(8Hz), **83**(12Hz), and **63**(16Hz).
   |fast32 - All |fast16 values plus **42**(24Hz) and **31**(32Hz).

   Note that fast measurement periods may be supported in some modes but not others, depending on the logger's configuration.

   ⚠ The following parameters are available only if the logger is configured to support at least one of the **average**, **burst**, **tide**, or **wave** modes.

4. **burstinterval** [=<*burstinterval*>] reports or sets the time between the first measurement of two consecutive bursts; it is not the gap between the end of one burst and the start of the next. The <*burstinterval*> is specified in milliseconds.

   The absolute limits of the permitted range are 1000 (one second) to 86400000 (24 hours), and the <*burstinterval*> must be set to a multiple of 1000. However, before the logger can be enabled for sampling the value set must also be consistent with the measurement **period** and **burstlength**.

5. **burstlength** [=<*burstlength*>] reports or sets the number of measurements taken in each burst.
   The permitted range is 1 to 65535, but before the logger can be enabled for sampling the value set must also be consistent with the measurement **period** and **burstinterval**.
   The constraining relationship between the burst parameters is: **burstinterval** > (**burstlength** * **period**).

   > ⚠ The following read-only parameter is available only if the logger is configured to support at least one of the gating conditions for sampling.

6. **gate** reports any gating condition currently enabled.
   A **gate** is an extra requirement that must be satisfied before sampling will occur, in addition to the logger's current time being between the start and the end times. The following gating conditions are presently defined:

   a. **none**: no gating conditions are enabled.
   b. **thresholding**: a threshold requirement must be satisfied; see the **thresholding** command.
   c. **twistactivation**: the end-cap must be in the "on" position; see the **twistactivation** command.
   d. **invalid**: two or more gating conditions have been selected, this is not currently possible and will prevent the logger from being enabled.

   For further information about gated sampling, see the section Gated Sampling (page 40).

   > ⚠ The following read-only parameter is available only if the logger is configured to support 'fast' sampling, in which the **period** can be less than 1000ms.

7. **userperiodlimit** reports the minimum period which can be used in 'fast' sampling modes, in milliseconds; the **period** can not be set to a value less than this.

Examples

```
>> sampling
<< sampling schedule = 1, mode = continuous, period = 167, burstlength = 60,
burstinterval = 300000, gate = none
```

The logger has been programmed for continuous 6Hz sampling.The programmed values of the burst parameters are reported but do not apply to continuous sampling. No gating condition is in force.

```
>> sampling mode = average
<< sampling mode = average
>> sampling
<< sampling schedule = 1, mode = average, period = 167, burstlength = 60, burstinterval =
300000, gate = none
```

Averaging enabled without changing the other parameters; the logger is now programmed to take a burst of 60 measurements at 6Hz every five minutes, and store the average of the 60 measurements.

```
>> sampling mode = burst, burstinterval = 600000
<< sampling mode = burst, burstinterval = 600000
>> sampling
<< sampling schedule = 1, mode = burst, period = 167, burstlength = 60, burstinterval =
600000, gate = none
```

The mode is changed to burst recording and the burst interval to ten minutes; the logger is now programmed to take a burst of 60 measurements at 6Hz every ten minutes, storing all measurements in memory.

```
>> sampling
<< sampling schedule = 1, mode = continuous, period = 250
```

The logger has been programmed for continuous 4Hz sampling. This logger does not support any of the other modes, so the parameters are not reported.

Errors

Error E0105
Settings may not be modified while logging is in progress; reading them is permitted.

Error E0108
The command was given with an argument which is unrecognized or has an invalid value; for example "sampling mode = mean", or "sampling schedule = 2".

Error E0109
An attempt was made to use a feature which the logger is not configured to support; for example "sampling mode = average" if the logger does not support averaging.

3.2 Gated Sampling

Gated sampling is available with loggers which are configured to support it. Normally, a logger will sample at the programmed period or interval when its current time is between the start time and the end time. If a gating condition is specified, then this further requirement must also be met before sampling will occur.

The start time and end time may still be used with gated sampling to define when the logger will examine the gating condition to see whether or not it should be sampling, but the logger will not sample if its current time is outside this range, no matter what state the gating condition is in. To have sampling activity depend *only* on the gating condition, the start and end times should be set to the extremes of the logger's date/time range: these are respectively 2000/01/01 00:00:00 and 2099/12/31 23:59:59. Other combinations are possible; for example a valid start time in the future may be set to ensure sampling is not started before then if the gating condition is inadvertently satisfied, while the end time is set to the maximum limit so that only the gating condition will stop the logger once sampling has started.

If a gating condition has been selected, it will be reported as the **gate** parameter in response to the **sampling** command. If sampling is paused by a gating condition, but the logger would otherwise be sampling and recording data, it will respond to the **status** command with the value **gated**.

When the logger's status is **gated**, it responds to commands in some instances as if it were **logging**. For example, schedule parameters may not be modified while the logger is enabled for sampling, and this includes the **gated** condition.

The following gating conditions are presently available, if the logger is configured to support them.

1. Thresholding.
2. Twist Activation.

3.2.1 thresholding

Usage

>> thresholding [<parameter> [= <value>], …]

Security

Open, Unsafe (modifications not permitted while logging).

Description

Reports or sets the parameters which control threshold gated sampling. The <*parameter*>s currently supported are:

1. **state** [**= on** | **off**] enables or disables the threshold gating feature.
2. **channel** [= < *index* >] the index of the channel to use for the threshold check. This must be a channel that exists in the logger; the first channel has an < *index* > of 1. The channel must also have a valid calibration, because the comparison of data readings with the threshold value is made in calibrated units.
3. **condition** [**= above** | **below**] specifies whether sampling will occur when the monitored parameter is above the specified threshold value, or below it.
4. **value** [= < *value* >] specifies the threshold value in calibrated units, given as a number compatible with floating point formats.
5. **interval** [= < *milliseconds* >] specifies the interval between threshold checks. The value is given and reported in milliseconds, but must correspond to a non-zero whole number of seconds, and not be greater than 24 hours (86400000ms).

The basic validity checks on parameters indicated above are performed immediately. Further checks may result in error messages at the time the logger is enabled; for example, setting a threshold check **interval** shorter than the logger can achieve in its programmed operating mode is not permitted.

Threshold-gated sampling is used to turn recording of data on or off, depending on the data value monitored from the selected channel.

When data recording is on, the logger is in the **logging** state. Readings from all channels are acquired, stored and otherwise processed according to the logger's programmed settings and schedule parameters. When data recording is off, the logger is in the **gated** state. Readings from only the selected **channel** are taken at the specified **interval**, and compared to the threshold **value**; they are not stored or otherwise processed.

When a logger is enabled with thresholding activated, it will initially assume the **gated** state if the start time has already passed. If the start time is in the future, the logger will initially be in the **pending** state, and move to the **gated** state when the start time is reached.

When in the **gated** state, if the reading from the selected **channel** satisfies the programmed **condition** (above or below) when compared to the threshold **value**, the logger will move to the **logging** state and begin to acquire and record data normally.

In most situations normal sampling will begin immediately, but it is possible for there to be a delay. For example, if the thresholding **interval** were 15 seconds, the sampling **period** 10 seconds, and a transition occurred at a time hh:mm:15 , the next scheduled sample would not be due until hh:mm:20 , so there would be a 5 second delay.

When in the **logging** state, the reading from the thresholding channel is still monitored at every sample time. If it no longer satisfies the programmed condition, the logger will move back to the **gated** state. However, this does *not* happen immediately. The logger remains in the **logging** state for a guard time of 10 seconds after the thresholding condition first fails; only if readings from the thresholding channel fail to satisfy the condition for 10 seconds continuously does the logger actually move to the **gated** state. If at any point during the 10 second guard time the condition is once again satisfied, the **logging** state persists and the guard time is reset, awaiting another possible transition.

This behaviour prevents data acquisition being interrupted by short episodes during which the thresholding condition is not satisfied.

Examples

```
>> thresholding
<< thresholding state = off, channel = 1, condition = above, value = 20.0000, interval = 15000
```

Threshold gating is not enabled; if it were, the logger would check the value read from Channel 1 every 15 seconds, and start normal sampling if above 20.0.

```
>> thresholding interval = 10000, value = 17.5
<< thresholding value = 17.5000, interval = 10000
```

The gating parameters are changed so that the logger would check Channel 1 every 10 seconds, and start normal sampling if the value were above 17.5.

```
>> thresholding state = on
<< thresholding state = on
```

Threshold gated control of sampling is enabled.

Errors

Error E0109
The logger is not configured to support threshold gated control of sampling.

Error E0105
Settings may not be modified while logging is in progress; reading them is permitted.

Error E0108
The command was given with an argument which is unrecognized or has an invalid value; for example "thresholding state = enabled", or "thresholding interval = 10".

Error E0601
The channel selected for threshold checking is not calibrated.

3.2.2 twistactivation

Usage

>> twistactivation [<parameter> [= <value>], …]

Security

Open, Unsafe (modifications not permitted while logging).

Description

Reports or sets the parameters which control twistactivation gated sampling. The <*parameter*>s currently supported are:

1. **state** [**= on** | **off**] enables or disables the twistactivation gating feature.
2. **location** reflects the current position of the endcap. This parameter can not be set.

The basic validity checks on parameters indicated above are performed immediately. Further checks may result in error messages at the time the logger is enabled; for example, enable thresholding and twistactivation gating features at the same are not permitted.

Twistactivation-gated sampling is used to turn recording of data on or off, depending on the position of the endcap on the instrument. This feature works only with compatible instruments.

Examples

```
>> twistactivation
<< twistactivation state = off, location = on
```

Twistactivation gating is not enabled; if it were, the logger would sample data depending on the position of the endcap.

```
>> twistactivation state = on
<< twistactivation state = on, location = on
```

Twistactivation gated control of sampling is enabled.

Errors

Error E0105
Settings may not be modified while logging is in progress; reading them is permitted.

Error E0108
The command was given with an argument which is unrecognized or has an invalid value; for example "twistactivation state = enabled".

Error E0109
The logger is not configured to support twistactivation gated control of sampling.

3.3 Vehicle support

3.3.1 regimes

Usage

>> regimes [direction | count | reference]

Security

Open, Unsafe (modifications not permitted while logging).

Description

Sets and returns information about the regimes. These are only used if the **sampling mode = regimes** .This is intended for use when the instrument is integrated into moving platforms.

1. **direction[= ascending | descending]** indicates if the instrument is intended to ascend or descend in the water column. The **boundary** values specified for each individual **regime** are the *first* boundary in the regime; so if the instrument is ascending, the boundary specified is the *lower* boundary; and vice-versa.
2. **count [= <countvalue>]** indicates the number of regimes that are set. Reducing this value will erase **regime** settings immediately. Increasing this value will add a **regime** and assign default values to the settings. The maximum number of regimes is 3.
3. **reference[= absolute | seapressure]** indicates which pressure is used as reference for the determination of the current regime and bin. The sea pressure is the difference between the absolute pressure measured and the atmospheric pressure setting defined via the command **settings**.

Examples

```
>> regimes
<< regimes direction = ascending, count = 3, reference = absolute
```

There are three specified regimes with the individual boundary values being interpreted as the lower values as the instrument ascends through the water column.

Errors

Error E0105
Settings may not be modified while logging is in progress; reading them is permitted.

Error E0108
The command was given with an argument which is unrecognized or has an invalid value; for example "regimes direction = horizontal", or "regimes count = 125".

Error E0109
An attempt was made to use a feature which the logger is not configured to support.

3.3.2 regime

Usage

>> regime <index> [boundary | binsize | samplingperiod]

Security

Open, Unsafe (modifications not permitted while logging).

Description

Returns information about the specified <*index*> regime. The first regime has an <*index*> of 1. If **direction** (regimes (page 44) command) is set to descending, the first regime corresponds to the regime the closest to the surface. If **direction** is set to ascending, the first regime is the closest to the seabed. Depending on how is set **reference** (regimes (page 44) command), the logger use the absolute pressure or the sea pressure to determine the current regime and the current bin.

The following parameters give the basic information available for all regimes.

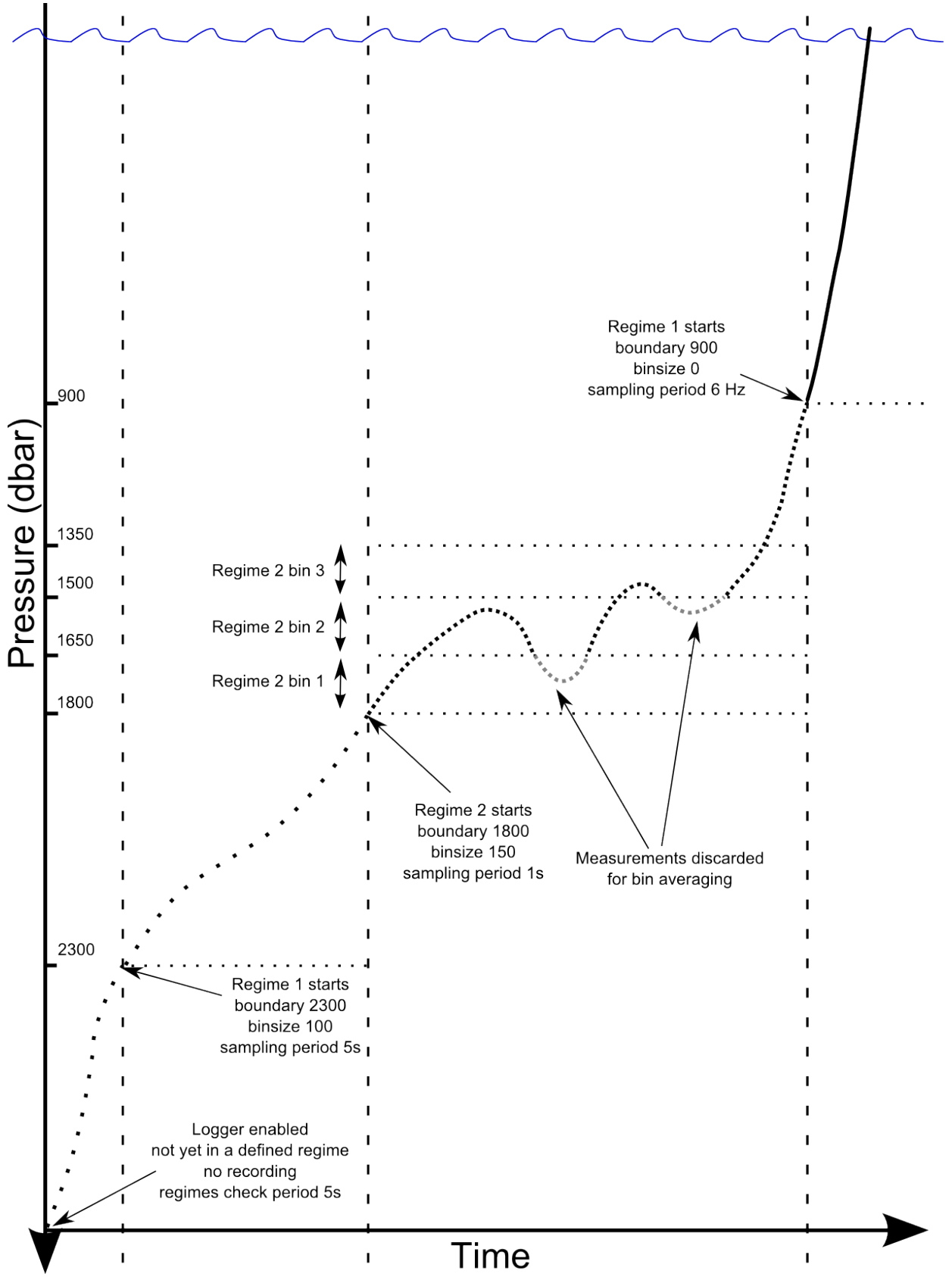
1. **boundary** [=<*firstboundaryvalue*>] specifies the transition from one regime to the next, and is interpreted as the *first* boundary in a region. The units are in dbar. The minimum precision is 1 dBar, and the value should be between 0 dbar and 65535 dbar.
2. **binsize** [=<*binvalue*>] specifies the size (in dbar) used for each averaged bin. This is typically set by the user to be a denominator of the total regime size, but if the last bin in the regime is smaller than the rest, the measurement is stored early and the next regime commences. The minimum precision is 0.1dBar, and the value should be between 0.1 dBar and 6553.5 dBar. If the binsize is set to 0.0, the logger will not average the data per bin during the regime and will just record/stream every measurement.
3. **samplingperiod** [=<*period*>] has the same meaning as the **period** value in the sampling (page 35) command, but applies *only* to this particular **regime.** The maximum period is 65000 milliseconds.

If the sampling rate is different between two regimes, the logger can take up to 2 seconds to switch between those two regimes. During this time, no samples will be acquired.

When the vehicle is not in a defined regime, the logger will sample internally at the same rate as the first regime in order to check when it has entered this regime. Measurements acquired during this period will not be stored or streamed.

The average for each bin is stored as soon as the vehicle enters the next bin in the direction set by the regimes command. In the following figure the regime 2 bin 1 is stored as soon as the vehicle enters the the regime 2 bin 2. When the vehicle goes back into the defined regime 2 bin 1 range, the measurements acquired are then discarded but all the measurements acquired in the regime 2 bin 2 range are taken into consideration as long as the vehicle has not entered yet the regime 2 bin 3.

The bin average is calculated without any kind of interpolation. In the case of a binsize set to 0.0, there is no averaging whatsoever.


Pressure (dbar)
900
1350
1500
1650
1800
2300
Regime 1 starts
boundary 900
binsize 0
sampling period 6 Hz
Regime 2 bin 3
Regime 2 bin 2
Regime 2 bin 1
Regime 2 starts
boundary 1800
binsize 150
sampling period 1s
Measurements discarded
for bin averaging
Regime 1 starts
boundary 2300
binsize 100
sampling period 5s
Logger enabled
not yet in a defined regime
no recording
regimes check period 5s
Time

Examples

```
>> regime 1
<< regime 1 boundary = 2000, binsize = 100.0, samplingperiod = 1000
```

Assuming the instrument is ascending, this indicates that the regime should commence as the sensor passes upwards through 2000dbar, and that bins inside the regime should be 100dbar in size. For each bin, an average of all samples will be made, with the sampling period of the instrument set to 1000ms. This regime will continue until another comes into force.

```
>> regime 3 boundary = 1000, binsize = 5.1, samplingperiod = 500
```

This sets the third regime to commence at 1000dbar, with a bin size of 5.1dbar and a sampling rate of 2Hz (500ms). As there are no other regimes defined, this one has no second boundary, so the instrument would continue measuring indefinitely and will only stop when the endtime is reached, or the stop (page 66) command is issued.

```
>> regime 3
<< regime 3 boundary = 20, binsize = 0.0, samplingperiod = 167
```

Assuming the instrument is ascending, this indicates that the regime should commence as the sensor passes upwards through 20 dbar, and that during this regime, every measurements will be recorded (sampled at 6Hz). As there are no other regimes defined, this one has no second boundary, so the instrument would continue measuring indefinitely and will only stop when the endtime is reached, or the stop (page 66) command is issued.

Errors

Error E0105
Settings may not be modified while logging is in progress; reading them is permitted.

Error E0107
An argument expected by the logger was not given with the command; for example, there must always be an <*index*> argument.

Error E0108
The command was given with an argument which is unrecognized or has an invalid value; for example "regime 1 boundary = -45", or "regime 5".

Error E0109
An attempt was made to use a feature which the logger is not configured to support.

3.3.3 ddsampling

Usage

>> ddsampling [direction | fastperiod | slowperiod | fastthreshold | slowthreshold]

Security

Open, Unsafe (modifications not permitted while logging).

Description

Sets and returns information about the directional dependent sampling mode. These settings are used only if the **sampling mode = ddsampling**. This mode is intended for use when the instrument is integrated into a moving platform.

1. **direction[= ascending | descending]** indicates when the instrument is sampling at the fast rate.
2. **fastperiod [= < *period* >]** has the same meaning as the period value in the sampling (page 35) command, but applies only when the logger detects that it is moving in the preferred **direction**. This must be shorter than the provided **slowperiod** parameter.
3. **slowperiod[= < *period* >]** has the same meaning as the period value in the sampling (page 35) command, but applies only when the logger detects that it is not moving in the preferred **direction**. This must be longer than the provided **fastperiod** parameter.
4. **fastthreshold[= *<dbar>*]** sets the boundary, based on the previous profile, where the logger should switch to the fast period sampling. The minimum precision is 0.1dBar and value should be greater than 0.
5. **slowthreshold[= *<dbar>*]** sets the boundary, based on the current profile , where the logger should switch to the slow period sampling. The minimum precision is 0.1dBar and value should be greater than 0.

The directional dependent sampling mode relies on the logger profiles detection scheme (see Profile detection events generation (page 138)). Therefore, it is intended only for vehicles that profile overall pressure changes greater than 3 dbar.

If the direction is set to ascending and an upcast is detected while sampling at the slow period, the logger will revert to the fast sampling period even if the threshold has not been crossed (and *vice versa* if the direction is set to descending). See the following example.

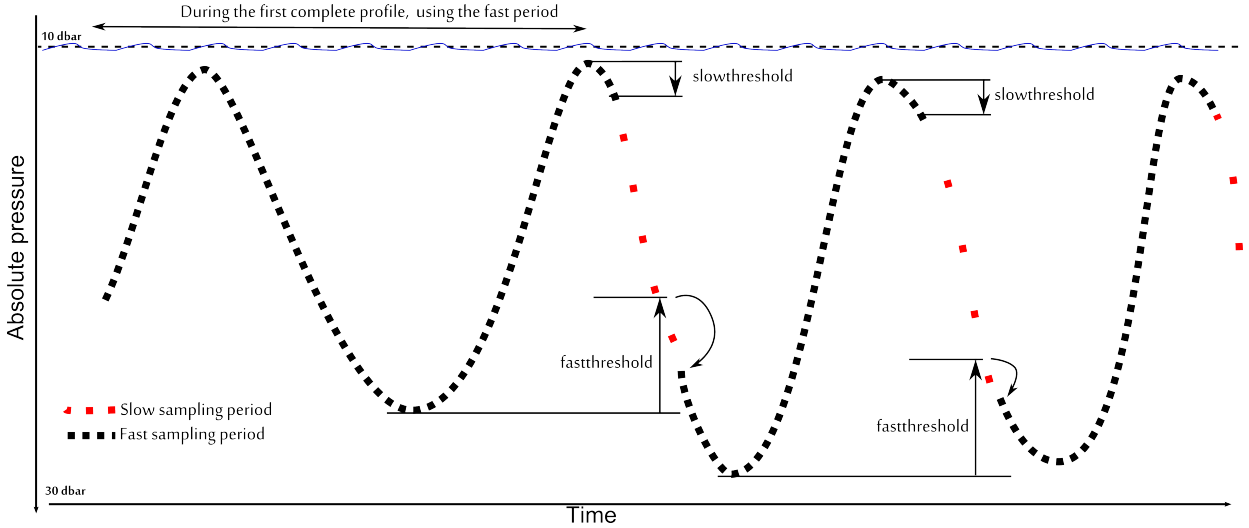
In order to configure the logger to maximize deployment life without compromising data in the preferred direction, the following guidelines can be used:

- **fastthreshold** should be greater than maxspeed x (**slowperiod** + modetransitiontime) + maxprofilevariation where:
  - modetransitiontime is the sum of latency and readtime as reported by the channels (page 77) command, plus one second (blanking period while switching sampling rate).
  - maxspeed is the maximum speed of the vehicle along the depth axis.
  - maxprofilevariation is the maximum variation in dbar from one profile to another.
- **slowthreshold** should be greater than the looping effect the vehicle might endure during its ascent (or descent).

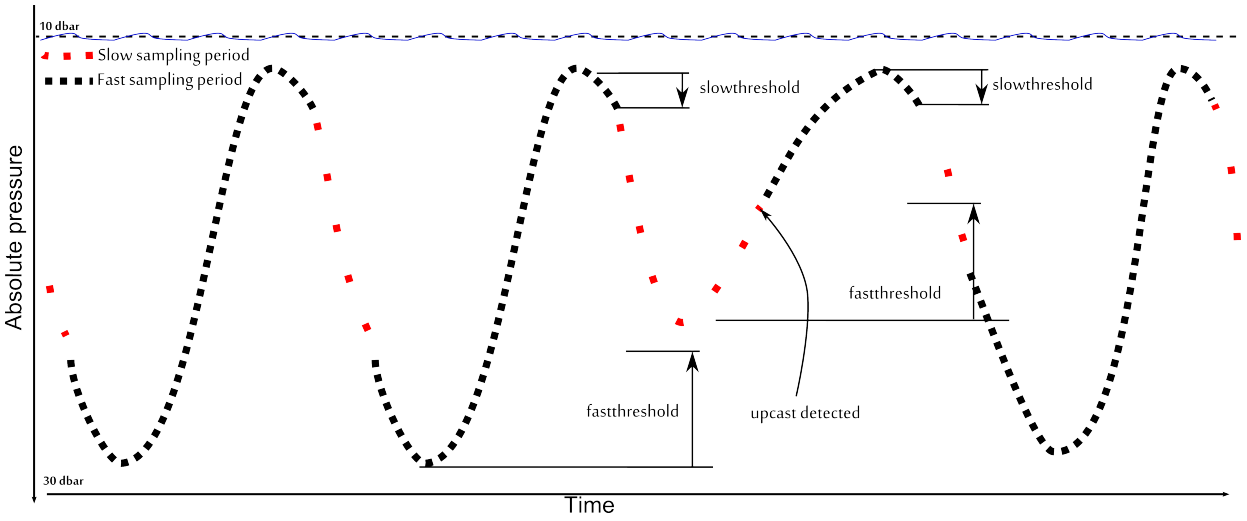
Examples

```
>> ddsampling
<< ddsampling direction = ascending, fastperiod = 167, slowperiod = 1000, fastthreshold = 5.0, slowthreshold = 1.6
```

The following picture gives a global overview of how the logger would behave with the previous example on a 20 dbar profile setup after it has been started.



The next picture shows how the directional dependent sampling would behave in the case of a profile much shorter than the previous one. In such a case the logger would detect an upcast while slow sampling and would then go back to the fast sampling mode.



Errors

Error E0105
Settings may not be modified while logging is in progress; reading them is permitted.

Error E0108
The command was given with an argument which is unrecognized or has an invalid value; for example "ddsampling direction = horizontal", or "ddsampling fastthreshold= 0.0".

Error E0109
An attempt was made to use a feature which the logger is not configured to support.

3.4 Real time data

3.4.1 outputformat

Usage

>> outputformat [channelslist] [support] [type [= <format-type>]]

Security

Open, Unsafe (modifications not permitted while logging).

Description

Reports or sets the format used to transmit data over the communications link; this format applies to both 'Fetched' data, and live 'Streamed' data if available. If no arguments are given, the current setting of the **type** parameter is reported.

The parameters currently supported are:

1. **channelslist** This reports a list of names and units for the active channels, in order. This list is helpful in identifying the channel corresponding to each value in the transmitted data. Any channels which have been turned **off** are excluded. The list will expand as support is added for more sensor types, so not all firmware versions will support every sensor type listed. If there is more than one channel of a particular type, the same information is reported for all of them.
   The list of all possible names is given below in alphabetical order. Some are quite generic, others are very sensor specific :
   - backscatter
   - BTEX
   - calphase
   - CDOM
   - chlorophylll
   - conductivity
   - crude_oil
   - custom_fluorometer
   - cyanobacteria
   - depth
   - dissolved-02
   - fluoroescein
   - fluorometry-UV
   - measurement_count
   - methane
   - optical_brighteners
   - ORP
   - PAR
   - partial CO2 pressure

- period
- pH
- phycocyanin
- phycoerythrin
- pressure
- pressure_period
- refined_fuels
- rhodamine
- salinity
- specific_conductivity
- speed of sound
- temperature
- temperature_period
- transmittance
- turbidity
- voltage

2. **support** reports the formats which are available.
   At present, the formats which may be supported are:

   a. **caltext01**, Calibrated ASCII output.
      The output starts with a timestamp giving years, months, days, hours, minutes, seconds and thousandths (milliseconds), punctuated as shown. A value for each channel is then sent; this is the measured parameter after conversion to physical units according to the instrument's current calibration. All values are shown to 4 decimal places. A comma and space separate the timestamp and values.
      Format:
      YYYY-MM-DD hh:mm:ss.ttt, <value1>, <value2>, ...

      Example for a 3-channel logger:
      2012-09-10 11:24:14.000, 38.6664, 21.5183, 10.9601

   b. **caltext02**, Calibrated ASCII output.
      The output starts with a timestamp giving years, months, days, hours, minutes, seconds and thousandths (milliseconds), punctuated as shown. A value for each channel is then sent; this is the measured parameter after conversion to physical units according to the instrument's current calibration. All values are shown to 4 decimal places. Following the value, and separated from it by a space, is a short

string representing the units of measurement. A comma and space separate the timestamp, and the information for each channel.
Format:
YYYY-MM-DD hh:mm:ss.ttt, <value1 units1>, <value2 units2>, ...

Example for a 3-channel logger CTD logger:
2012-09-10 11:52:21.000, 38.6671 mS/cm, 22.0217 C, 10.9596 dBar

c. **caltext03**, Calibrated ASCII output.
The output starts with a timestamp giving years, months, days, hours, minutes, seconds and thousandths (milliseconds), punctuated as shown. A value for each channel is then sent; this is the measured parameter after conversion to physical units according to the instrument's current calibration. All values are shown with enough significant digits ensuring no loss of resolution. A comma and space separate the timestamp and values.
Format:
YYYY-MM-DD hh:mm:ss.ttt, <value1>, <value2>, ...

Example for a 3-channel logger CTD logger:
2012-09-10 11:52:21.000, 38.6671142, 22.0217241, 10.9596633

d. **caltext04**, Calibrated ASCII output.
The output starts with a timestamp giving years, months, days, hours, minutes, seconds and thousandths (milliseconds), punctuated as shown. A value for each channel is then sent; this is the measured parameter after conversion to physical units according to the instrument's current calibration. All values are shown in 'engineering-notation' floating point (same as scientific notation except that the exponents are constrained to be multiples of three) with enough significant digits ensuring no loss of resolution. A comma and space separate the timestamp and values.
Format:
YYYY-MM-DD hh:mm:ss.ttt, <value1>, <value2>, ...

Example for a 3-channel logger CTD logger:
2012-09-10 11:52:21.000, 38.6671142e+000, 22.0217124e+000, 1.95962418 e+003

3. **type** [= caltext01 | caltext02 | caltext03 | caltext04] set and/or report the current output format type.
Not all instruments support all the formats; use the **support** option to check which formats are available.

Note that for all the **caltext** formats, any individual channel value may be replaced by one of the following:

- **Error-**<*EC*>: an identifiable error occurred on this channel; for a list of the possible 2-digit error codes <*EC*>, see the paragraph 'Error Codes' in the Section 'Sample data standard format (page 131)'.
- **nan**: the value is Not A Number in IEEE floating point format, which indicates an internal problem with calculating the value.
- **inf** / **-inf**: attempting to calculate the value produced a result outside the range which can be represented.
- ###: the channel is not calibrated, so an output value could not be calculated.

Examples

```
>> outputformat
<< outputformat type = caltext01
```

```
>> outputformat channelslist
<< outputformat channelslist = temperature (C), pressure (dbar)
```

```
>> outputformat support
<< outputformat support = caltext01, caltext02, caltext03, caltext04
```

```
>> outputformat type = caltext02
<< outputformat type = caltext02
```

Errors

Error E0108
The command was given with an argument which is unrecognized or misplaced.

3.4.2 streamusb

Usage

>> streamusb [state [= on | off]]

Security

Open

Description

Changing the state of USB streaming while logging is in progress is permitted, but will cause a time-stamped event to be recorded in memory with the sample data.

Examples

```
>>streamusb
<<streamusb state = on
```

```
>>streamusb state = off
<<streamusb state = off
```

Errors

Error E0108
The supplied argument was not valid or out of place.

Error E0109
The logger is not configured to support streaming on the USB link.

3.4.3 streamserial

Usage

>> streamserial [state [= on | off]] [aux1_all] [aux1_state [= on | off]] [aux1_setup [= <*setup_time*>]] [aux1_hold [= <*hold_time*>]] [aux1_active [= high | low]] [aux1_sleep [= high | low | tristate]]

Security

Open

Description

If the logger is configured to support streamed data over the serial link, this command can turn the feature on or off, and also configure the behaviour of an auxiliary RS-232 control output, AUX1. If the command is given with no parameters, the value of the **state** parameter is reported. The operating parameters of the serial link, such as baud rate and mode, are accessed using the serial (page 93) command.

state [= on | off]: reports the state of the streamed data feature, and optionally turns it on or off. When the feature is on, acquired data is sent over the serial link at the same time it is stored in memory. Refer to the outputformat (page 52) command for descriptions of the data formats which may be available. When the feature is off, data is not sent. Changing the state of serial streaming while logging is in progress is permitted, but will cause a time-stamped event to be recorded in memory with the sample data.

The **aux1**_... options configure the behaviour of an auxiliary RS-232 output signal AUX1, if the logger is configured to support it. This signal is available only if the mode of the Serial link is set to RS-232; refer to the serial (page 93) command for details.

The signal can be used to control an external device such as a modem, but is intended only for the purpose of transmitting streamed data. The logger may activate the signal during other transmissions on the serial link, but it is intended only for enabling a transmitting device for remote monitoring of send-data-only installations. It is not intended to be, and should not be used as, a general purpose flow control signal. Below are the descriptions of the individual command parameters.

aux1_all: a read-only option which causes the values of all the other **aux1_...** parameters to be reported.

aux1_state [= on | off]]: reports the on/off state of the feature, or optionally enables or disables the feature as required. When the feature is disabled, the remaining **aux1_...** parameters have no effect. The default setting as shipped from the factory is **off**.

aux1_setup [= <*setup_time*>**]**: reports or sets the AUX1 signal set-up time, in milliseconds. When the logger is sampling and is about to stream data over the serial link, this is the time for which AUX1 will be set to the active level before the streaming transmission begins. The valid range of values is 10...120000 (10ms to 2 minutes); the default value as shipped from the factory is 1000ms.

aux1_hold [= <*hold_time*>**]**: reports or sets the AUX1 signal hold time, in milliseconds. This is the time for which AUX1 will be held at the active level after the serial streaming transmission has finished. The valid range of values is 10...120000 (10ms to 2 minutes); the default value as shipped from the factory is 1000ms.

aux1_active [= high | low]: reports or sets the active level of the AUX1 signal seen by the external device during the setup time, data transmission and hold time. The high and low signal levels are approximately +5V and –5V respectively, compatible with the RS-232 specification. The default setting as shipped from the factory is **high**.

aux1_sleep [= high | low | tristate]: reports or sets the level of the AUX1 signal seen by the external device while the logger is asleep. In the **high** and **low** states the signal is actively driven to the appropriate level by the logger, which may be necessary for some external devices. The high and low signal levels are approximately +5V and –5V respectively, compatible with the RS-232 specification. However, these two options cause a large increase in

the logger's sleep current, and will severely impact the available deployment lifetime when using the logger's internal batteries. These options are not recommended for use unless the logger is run from an external power source for which a high sleep current does not matter. In the **tristate** condition, the signal is not actively driven, but becomes high impedance. This allows the logger to maintain a very low sleep current, but the external device must be able to enter the appropriate 'off' or 'sleep' state under these conditions. The default setting as shipped from the factory is **tristate**.

Examples

```
>> streamserial
<< streamserial state = off

>> streamserial state = on
<< streamserial state = on
```

```
>> streamserial aux1_all
<< streamserial aux1_state = on, aux1_setup = 2000, aux1_hold = 3000, aux1_active = high, aux1_sleep = tristate

>> streamserial aux1_setup = 500
<< streamserial aux1_setup = 500
```

Errors

Error E0108
The supplied argument was not valid or out of place.

Error E0109
The logger is not configured to support streaming over the serial link.

3.5 Deployment

3.5.1 status

Usage

>> status

Security

Open.

Description

Returns the current state of the finite state machine for the instrument's logging function. Possible states are given below:

- **disabled** : logging is not enabled.
- **pending** : logging is enabled but before the start time.
- **logging** : logging is in progress.
- **gated** : logging paused, waiting for a gating condition to be satisfied.
- **finished** : the programmed end time has been passed.
- **stopped** : a **stop** command was received.
- **fullandstopped** : memory full, logging has stopped.
- **full** : memory full, logger continues to stream data.
- **failed** : stopped, internal error.
- **notblank** : memory failed to erase.
- **unknown** : internal error, state unknown.

Most of these are self explanatory. In the very unlikely event that they occur at all, the states **failed**, **notblank** and **unknown** arise from serious internal errors, and ideally the instrument should be returned to RBR Ltd for further analysis. However, if it would be preferable to attempt deployment anyway, try these recover procedures.

1. Send a **permit memclear** and **memclear** sequence (see the memclear (page 74) command), which may succeed in erasing the memory and resetting the instrument status to **disabled**. It may then be possible to continue using the logger and to enable it for a new deployment.
2. Send a **reboot** command, using the delay parameter if communicating over a USB link (see the reboot (page 109) command), then try (1) above.
3. Remove all batteries, power, and USB connections from the logger for at least five minutes. Replace the batteries or apply power, set the correct date and time (see the clock (page 31) command), then try (1) above.

If the **failed** status resulted from an attempt to enable the logger, then either it failed to store a deployment header in memory, or the alarm time for the next sample could not be correctly loaded into the Real Time Clock/Calendar. If the logger does have a fault, the second case could happen again at any time during a deployment, so there is a risk that the logger may fail again and the deployment will terminate early.

In any event the instrument should be returned to RBR Ltd for further analysis at the earliest available opportunity, with as much detail as possible about the circumstances of the failure.

Examples

```
>> status
<< status = stopped
```

```
>> status
<< status = logging
```

Errors

None.

3.5.2 verify

Usage

>> verify

Security

Open.

Description

This command performs all the same schedule consistency checks which the **enable** command performs. It then reports the same response which the **enable** command would produce, whether this is an updated logger status, a warning or an error message. It does not, however, actually enable the logger for sampling.

In other words, it performs a "dry run" of the **enable** command to allow the programmed schedule parameters to be verified.

> verify always checks that the data memory is empty.

Examples

```
>> verify
<< verify = pending
```

The programmed schedule is valid and the logger would, if enabled, begin collecting data at the start time.

```
>> verify
<< verify = logging
```

The programmed schedule is valid and the logger would, if enabled, begin collecting data at the next scheduled sample time.

Warning messages:
E0401 estimated memory usage exceeds capacity
The schedule is valid, but the memory will fill up before the end time is reached.

Errors

Error E0402
The data memory must be empty.

Error E0403
The end time must be later than the start time.

Error E0404
The end time must be later than the logger's current time.

Error E0408
The logger must not already be enabled or active in any sense.

Error E0409
The runtime error log must be clear.

Error E0410
All the logger's measured channels have been turned off (in instruments which support this feature); there is nothing to sample.

Error E0411
The measurement period is not consistent with other programmed parameters.

Error E0412
The programmed burst parameters are not consistent for the selected sample mode.

Error E0413
Serial streaming may not be available at high sample rates.

Error E0414
The requested threshold check interval is not consistent with the logger's capabilities or other programmed parameters.

Error E0415
Gating conditions such as thresholding and twist activation can not be used in combination; choose one.

Error E0416
The programmed settings for 'regimes' sampling are not consistent.

Error E0417
A gating condition can not be used with the 'regimes' sampling mode.

Error E0418
The cast detection feature can not be used if the logger has no pressure or depth channel.

Error E0419
Valid calibration coefficients for all sampled channels must be present before logging can be enabled.

Error E0420
The indicated channel has been turned off, but it must be sampled for a selected feature (eg. thresholding) to work; turn the channel back on.

Error E0421
Raw streaming output formats are not permitted if the memory storage format is set to EasyParse (calbin00).

Error E0422
The AUX1 signal can be used only in RS-232 mode.

Error E0423
The programmed settings for 'ddsampling' sampling are not consistent.

3.5.3 enable

Usage

>> enable [erasememory = true]

Security

Open.

Description

Enables a logger to sample according to the programmed schedule.
An option is available to erase the memory and enable the logger with a single command, otherwise the **memclear** command must be used to erase the memory beforehand if necessary.

Although the enable command is always available, a number of checks are made before logging is actually enabled. If any check fails, the logger is not enabled and an error message is sent as described below. The most severe error found causes immediate failure of the command; a single attempt to enable the logger will not detect multiple errors.

If all checks are passed, the logger's status will be reported as one of:
- **pending**, if logging is enabled but the start time is still in the future,
- **logging**, if logging is now in progress, or
- **gated**, if logging would now be in progress but the selected gating condition is not satisfied.

If the memory would fill before the end time is reached, a warning message is issued but the logger is still enabled to sample.

If all required conditions are satisfied, an internal error may still prevent sampling from being enabled when the logger attempts it. This also will provoke an error message.

Examples

```
>> enable
<< enable = pending
```

The programmed schedule is valid and the logger will begin collecting data at the start time.

```
>> enable
<< enable = logging
```

The programmed schedule is valid and the logger will begin collecting data at the next scheduled sample time.

```
>> enable
<< E0401 estimated memory usage exceeds capacity, enable = pending
```

The programmed schedule is valid and the logger will begin collecting data at the start time; however, the memory may fill before the end time.

Warning messages:

E0401 estimated memory usage exceeds capacity
The schedule is valid, but the memory may fill up before the end time is reached.

Errors

Error E0107
For example, the **true** argument was not given with the **erasememory** option.

Error E0108
An unrecognized argument was given with the command.

Error E0301
Memory erase failed (requested with **erasememory** = **true** option).

Error E0402
The data memory must be empty

Error E0403
The end time must be later than the start time.

Error E0404
The end time must be later than the logger's current time.

Error E0405
An internal problem prevented the logger from enabling the sampling schedule. For further details and suggestions for recovery, refer to the status (page 59) command.

Error E0408
The logger must not already be enabled or active in any sense.

Error E0409
The runtime error log must be clear.

Error E0410
All the logger's measured channels have been turned off (in instruments which support this feature); there is nothing to sample.

Error E0411
The measurement period is not consistent with other programmed parameters.

Error E0412
The programmed burst parameters are not consistent for the selected sample mode.

Error E0413
Serial streaming may not be available at high sample rates.

Error E0414
The requested threshold check interval is not consistent with the logger's capabilities or other programmed parameters.

Error E0415
Gating conditions such as thresholding and twist activation can not be used in combination; choose one.

Error E0416
The programmed settings for 'regimes' sampling are not consistent.

Error E0417
A gating condition can not be used with the 'regimes' sampling mode.

Error E0418
The cast detection feature can not be used if the logger has no pressure or depth channel.

Error E0419
Valid calibration coefficients for all sampled channels must be present before logging can be enabled.

Error E0420
The indicated channel has been turned off, but it must be sampled for a selected feature (eg. thresholding) to work; turn the channel back on.

Error E0421
Raw streaming output formats are not permitted if the memory storage format is set to EasyParse (calbin00).

Error E0422
The AUX1 signal can be used only in RS-232 mode.

Error E0423
The programmed settings for 'ddsampling' sampling are not consistent.

3.5.4 stop

Usage

>> disable

Security

Open.

Description

If the instrument is logging, this command will terminate the current deployment. If the instrument is not currently logging, a warning message is returned. The status of the logger is always reported when the command is complete.

If the **stop** command is sent while a measurement is in progress, the measurement will be completed before logging is stopped. Consequently, depending on the channels installed in the logger and the sampling mode, the logger's response to the command may be delayed. If the logger is sampling in any averaging or burst recording mode, the burst currently in progress will be interrupted and abandoned.

If the logger is recording data to memory, a 'stop event' will be appended to the data after the last sample stored.

For **fwtype = 103** only (see the id command), the alternate name **disable** may be used instead of **stop**, but for compatibility with all current instruments from RBR Ltd. the use of **stop** is recommended.

Examples

```
>> stop
<< stop = stopped
```

```
>> stop
<< E0406 not logging, stop = disabled
```

Warning messages:
E0406 not logging
The **stop** command was issued when the logger is not actually logging.

3.5.5 simulation

Usage

>> simulation [state [= on|off]] [period [= <*milliseconds*>]]

Security

Open for reading, Protected for update, Unsafe for update (modifications not permitted while logging).

Description

This command is available only in firmware versions 1.300 or later. When **state = on**, the measured values from selected channels in the logger are replaced by artificially generated values. These values follow an approximately linear ramp which travels up and down between predefined limits, taking **period = <*milliseconds*>** to complete one full cycle. All simulated

channels cycle at the same rate. The channel types supported and the limits which apply are listed below. If the logger has channels of other types, data for those channels is replaced by a constant value.

Channel type	Minimum	Maximum	Units
temperature	-5	+35	°C
pressure	+10	+2000 (1)	dbar
conductivity	-1	+85	mS / cm
PAR	-25	+2500	µmol / m^2 / s
turbidity	-25	+2500	NTU
chlorophyll	-2	+150	µg / L

Notes

1. If a pressure channel's calibration coefficients indicate that 2000dbar is beyond the measurement range, a limit corresponding to the sensor's maximum output will be used instead.

The simulated values are used both for scheduled samples stored in memory, and for on-demand samples obtained using the fetch (page 104) command. Both types of sample should confirm closely to the same linear ramp; if scheduled and on-demand samples are required simultaneously, there may be some small deviations due to the computation's attempts to satisfy both.

A logger programmed to generate simulated data can be identified in one of two ways: request the setting of the simulation state, or check the response to the id (page 97) command for the indicator as shown below.

```
>> simulation state
<< simulation state = on|off
>> id
<< id mode = SIMULATION, model = RBRconcerto, version = 1.300, serial = 012345,
fwtype = 103
    (Simulated data is enabled)
>> id
<< id model = RBRconcerto, version = 1.300, serial = 012345, fwtype = 103
```

```
    (Simulated data is disabled)
```

Examples

```
>> simulation
<< simulation state = off, period = 600000

>> permit simulation
<< permit = simulation
>> simulation state = on, period = 3600000
<< simulation state = on, period = 3600000

>> permit simulation
<< permit = simulation
>> simulation period = 1200000
<< simulation period = 1200000
```

Errors

Error E0105
Changes to the **state** or **period** parameters can not be made while logging is enabled.

Error E0103
Changes to the **state** or **period** parameters can not be made without issuing the command **permit simulation** immediately beforehand.

Error E0108
One of the supplied parameter names or values was not valid.

3.6 Memory and Data Retrieval

These commands provide information about the memory in which deployment data is stored, permit access to that data for retrieval, and allow the memory to be cleared.

3.6.1 meminfo

Usage

>> meminfo [dataset | used | remaining | capacity | size | all]

Security

Open.

Description

Reports information about the usage and characteristics of the data memory.

1. **dataset** is the index of the dataset being queried - see below for details.
2. **used** is the number of bytes actually used to store data in this dataset.
3. **remaining** is the number of bytes still available for data storage.
4. **size** is the maximum total size in bytes of the dataset.
5. **capacity** is the maximum number of bytes which will be stored in data memory if all of the 'remaining' bytes are fully used. This parameter is not reported by default, but must be requested explicitly.

The **dataset** argument is present to support the EasyParse data storage format, which assigns different types of deployment data to different datasets as follows:

1. the deployment header is in dataset 2,
2. the sample data is in dataset 1, and
3. events are in dataset 0.

In Standard data storage format, everything is in dataset 1. For a full discussion of data storage formats, refer to the section Format of Stored Data (page 111).

If the **dataset** argument is omitted then dataset 1 is assumed, and no <*dataset*> value is reported in the command's response, either: it is assumed that dataset 1 is implicitly understood. This is to minimize the risk of incompatibility with host software written to handle earlier firmware versions.

In Standard format, only one dataset is used, and so the remaining parameters have physical interpretations which are not too hard to understand. The **size** parameter is a fixed number for a given flash memory device. Because of the way these devices store data, it is sometimes not possible to use the entire device in order to reliably store small amounts of information. In such cases small areas of the device are not used to store data, but are no longer available for storage, either. Over the course of a deployment these uncommitted areas can accumulate, so (**used** + **remaining)** < **size**. However, (**used** + **remaining)** = **capacity** should always be true.

In EasyParse format, the relationship between the parameter values and the characteristics of the physical storage device become more complicated. For the curious, this is discussed in detail in the "Technical note" below, but this should not be considered required reading. Although the numbers will not be completely accurate, a reasonable guide to the state of the memory can be obtained by looking at the values for dataset 1 alone, since it contains the sample data, and there will typically be far more sample data than anything else.

Examples

```
>> meminfo
<< meminfo used = 1528, remaining = 134216192, size = 134217728
```

```
>> meminfo capacity
<< meminfo capacity = 134217720
```

```
>> meminfo dataset 1
<< meminfo dataset = 1, used = 1528, remaining = 134216192, size = 134217728
```

```
>> meminfo dataset 0 used
<< meminfo dataset = 0, used = 362
```

Errors

Error E0108
The supplied argument was not recognized.

Technical note
This note discusses the relationships between the parameter values for **used**, **remaining**, **size** and **capacity** when more than one dataset in the memory is in use; for example, when using the EasyParse data storage format.
To understand this, the concept of memory 'blocks' must be introduced. A block is the smallest amount of memory in the physical device which can be erased, and is relatively large; a typical size might be 128KB.
All datasets start off empty, and all blocks are initially available to any dataset. As soon as a dataset is written to, the modified block is assigned to that dataset. The memory within it is available to that dataset, but to no others, so for all the other datasets, the size of the memory appears to shrink by one block. If this strategy were not followed, datasets could not be erased independently of each other.
Therefore, as multiple datasets are opened and written to, each can report a different **size** for the memory, because as far as each dataset is concerned any blocks assigned to other datasets do not exist. Further, as blocks are assigned to each dataset, the **size** according to other datasets appears to change.
In fact, then, **size** is the maximum possible amount of memory which could be used by

the specified (or default) dataset, not the entire physical device. It is reported in bytes, but will always be a multiple of the block size. If only one dataset exists, then **size** does also give the physical device size, because no other datasets will use any blocks. Mercifully, **used** is a bit simpler to understand; it is just the number of bytes which have actually had data written to them in the specified dataset.
The value of **remaining** is the maximum possible number of bytes which could be written to this dataset in the future. It includes all bytes in unallocated blocks, as well as all available bytes in any partially used block assigned to this dataset.
Finally, **capacity** is the maximum possible number of bytes of data that the specified dataset could end up holding if all remaining bytes get used: in other words, (**used** + **remaining**). In general, **capacity** may be less than **size** for any data set, but never more. The reason for this is that sometimes a 'partial page' of data may be flushed from a RAM cache to the actual flash device before the cache is full. When this happens, the unused bytes in that page have no data in them, but they are no longer available for storage either; they are 'lost'. The accumulation of such lost bytes is the difference between **capacity** and **size**.

3.6.2 read

Usage

>> read <read-target> <target-parameter-list…> [<size-wanted>] [<from>]

Security

Open.

Description

This is a general purpose command which uses a binary transfer protocol to quickly retrieve relatively large quantities of information from the logger. The only area currently accessible by end users in this way is the data memory, although what constitutes the "data memory" will depend on the data storage format in use. For a full discussion of data storage formats, refer to the section Format of Stored Data (page 112).
For each instance of the **read** command:

1. there is a single transfer,
2. the data is CRC protected, and
3. arguments to the command control how much data is transferred and from where.

The command arguments are treated as follows:

1. <*read-target*> determines the nature of the information which will be read from the logger.
2. <*target-parameter-list…*> one or more target-specific parameters.
3. <*size-wanted*> determines the maximum number of bytes of data to be returned.
4. <*from*> determines the starting offset in the <*read-target*> area.

The response to the command has the form:
<< <*read-target*> <*target-parameter-list*…><*size-returned*> <*from*>**<cr><lf>**<*data*><*crc*>
The elements in the response are:

1. <*read-target*> and <*target-parameter-list…*> are simply the received arguments, echoed for confirmation.
2. <*size-returned*> gives the actual byte count of the <*data*> field; this may be less than <*size-wanted*>, but will never be greater.
3. <*from*> is the start address of the data, and is usually the same as given with the command. However the <*from*> command argument is optional; if not given, data will be read starting from the end point of the previous transfer, and the <*from*> address reported in the response will reflect this.
4. **<cr><lf>** ensures <*from*> is properly terminated in the event that the first byte of <*data*> happens to be a digit. It also allows the host to easily separate the acknowledgment to the **read** command from the actual <*data*>.
5. <*data*> is the actual binary data, delimited at the start by the **<cr><lf>** and bounded in length by <*size-returned*>.
6. <*crc*> is a 16-bit CRC using the CCITT polynomial f(x) = x^16 + x^12 + x^5 + 1, feeding bytes into the generator LSB first. If the bytes of the computed <*crc*> are swapped and appended to the data, the host can include them in its CRC-check as an extra two bytes: if the CRC is correct, this always gives a result of zero.

No acknowledge mechanism is implemented; failed transfers can simply be requested again.
As mentioned above, the only area presently accessible to end users via the **read** command is the data memory, for which:

1. <*read-target*> is **data**, and
2. <*target-parameter-list…*> is a dataset number.

For the Standard data storage format the dataset number is always 1; all deployment information is stored in a single dataset.
For the EasyParse format, the three major components of deployment data are each assigned a dataset as follows:

1. the deployment header is in dataset 2,

2. the sample data is in dataset 1, and
3. events are in dataset 0.

The **read** command knows nothing of the structure of the retrieved data in any of these cases, it is simply transferring a block of bytes of unknown content.
Internally, requests for very large amounts of **data** in a single instance of the **read** command are broken down into smaller blocks. Such a transfer could take a considerable time, and if necessary it may be aborted by sending the pseudo-command **abort**, followed by a <**cr**> character. The transfer then stops at the next internal block boundary, and terminates with the message **operation aborted**.

Examples

```
>> read data 1 1000 0
<< data 1 1000 0<cr><lf><bytes[0..999]-of-data><crc>

>> read data 1 1000 1000
<< data 1 1000 1000<cr><lf><bytes[1000..1999]-of-data><crc>

>> read data 1 1000 2000
<< data 1 12 2000<cr><lf><bytes[2000..2011]-of-data><crc>
```

There were 2012 bytes in dataset 1 of the memory in total. This could be determined in advance by **meminfo used**.

Errors

Error E0107
An argument expected by the logger was not given with the command; for example, there must always be a <read-target> argument.

Error E0108
The command was given with an argument which is unrecognized or has an invalid value.

3.6.3 memclear

Usage

>> memclear

Security

Protected, Unsafe (not permitted while logging).

Description

Clears the data storage area of the flash memory. Currently, all datasets are erased, regardless of the data storage format in use.

The logger responds by reporting that the memory used is zero if successful, as shown in the Example below; an error message is sent if the operation fails.

Examples

```
>> permit memclear
<< permit = memclear
>> memclear
<< memclear used = 0
```

Errors

Error E0103
permit memclear must immediately precede the command to clear the memory.

Error E0301
The memory failed to erase; if repeated attempts are not successful, please contact RBR Ltd for assistance.

3.6.4 memformat

Usage

>> memformat [support | type | newtype [= rawbin00 | calbin00]]

Security

Open, Unsafe (modifications not permitted while logging).

Description

This command reports or sets the format used to store deployment data in memory. The available parameters and their usage are described below.

1. **support** requests a list of the data storage formats available for this logger.
2. **type** requests the format of the data presently stored in memory, either for a deployment in progress or for one which has finished. If the memory is completely empty because it has been cleared, the response will be **none**.

3. **newtype** requests or sets the format of the data which will be used during a future deployment; the value can not be modified while logging is in progress.

If no argument is given to the command, the **type** is reported. Currently supported types are as follows:

rawbin00
This is the 'Standard' format which contains all sample data and supplementary data in a single data set in binary form.

calbin00
This is the 'EasyParse' format, for which the sample data is stored in its own data set in a uniform and consistent format which is simpler to decode.

For a full discussion of the data storage formats available, refer to the section Format of Stored Data (page 112).

Examples

```
>> memformat support
<< memformat support = rawbin00, calbin00
```

```
>> memformat
<< memformat type = rawbin00

>> permit memclear
<< permit = memclear
>> memclear
<< memclear used = 0

>> memformat
<< memformat type = none
```

```
>> memformat newtype = calbin00
<< memformat newtype = calbin00

>> memformat type
<< memformat type = none
```

Errors

Error E0105
The data storage format may not be modified while logging is in progress; reading is permitted.

Error E0108
The supplied argument was not recognized.

3.7 Configuration Information and Calibration

3.7.1 channels

Usage

>> channels [count | on | latency | readtime | minperiod | all]

Security

Open.

Description

A read-only command which returns general channel information for the logger.

1. **count** is simply the number of channels installed and configured in the logger.
2. **on** gives the number of active channels, which excludes any channels turned **off** by the user: see the **status** argument to the **channel** command.
3. **latency** is the power-on settling delay in milliseconds; this is the time the logger will wait after waking from its quiescent state, before attempting to take measurements. It allows all sensors and channel electronics to reach a stable condition. This overall latency delay is determined by the longest of all the individual *active* channel delays; channels which are turned **off** do not contribute.
4. **readtime** is the overall reading time in milliseconds; this is the additional time the logger needs to acquire data from all active channels once the latency delay has expired. This overall readtime is determined by the longest value of all the individual *active* channels; any which are turned **off** do not contribute.
5. Most channels have a fixed, pre-determined readtime, but for some it may be variable. An example would be a channel which supports, and is configured to use, the auto-ranging feature: the readtime is longer when the channel is in auto-ranging mode than when operated in a fixed-gain mode. The logger adjusts the reported value of the readtime to reflect the operating mode and status of all active channels.

6. **minperiod** is the minimum sampling period in milliseconds with the currently active channels. This takes account of the current overall values for latency and readtime, and adds some overhead and safety margin for both fixed and per-channel activities. The value applies to the "normal" sampling mode supported by all loggers; loggers configured to support fast sampling modes as well may use selected periods less than this value.

Examples

```
>> channels
<< channels count = 2, on = 2, latency = 160, readtime = 150, minperiod = 1000
```

```
>> channels latency readtime
<< channels latency = 160, readtime = 150
```

Errors

Error E0108
An unrecognized argument was given with the command.

Error E0505
Indicates a serious fault with the logger; please contact RBR Ltd for help.

3.7.2 channel

Usage

>> channel <index> [type | module | status [= on | off] | latency | readtime | equation | userunits | gain [= <gain-setting>] | gainsavailable | derived

Security

Open, Unsafe (modifications not permitted while logging).

Description

Returns information about the channel at the specified <*index*>: the first channel has an <*index*> of 1.

A special value of **all** may be given for the channel <*index*>, causing the requested parameters to be reported for all channels. The output for each channel is terminated by a vertical bar character '**|**', except for the last channel which is terminated by a <**cr**><**lf**> pair as normal. The following parameters give the basic information available for all channels. None of these values may be modified by end users.

1. **type** is a short, pre-defined 'generic' name for the installed channel; for example:
    a. **temp09** RBR*duo* temperature,
    b. **pres19** RBR*duo* pressure,
    c. **cond05** RBR*concerto* marine conductivity,
    and so on. RBR Ltd continually adds support for more sensor types and variants; the Section Supported Channel Types (page 139) contains a complete listing of channel types available at the time of writing this document.
2. **module** is the internal address to which this channel responds; it is normally of no interest to end users.
3. **latency** is the minimum power-on settling delay in milliseconds required by this channel, taking into account both the sensor and the interface electronics.
4. **readtime** is the typical data acquisition time in milliseconds required by this channel, again taking into account both the sensor and the interface electronics.
    Most channels have a fixed, pre-determined readtime, but for some it may be variable. An example would be a channel which supports, and is configured to use, the auto-ranging feature: the readtime is longer when the channel is in auto-ranging mode than when operated in a fixed-gain mode. The logger adjusts the reported value of the readtime to reflect the operating mode and status of the channel.
5. **equation** is the type of formula used to convert raw data readings to physical measurement units. The values for the core equations are shown below as examples; see the section Calibration Equations and Cross-channel Dependencies (page 148) for details of all supported equations.
    a. **tmp** temperature
    b. **lin** linear
    c. **qad** quadratic polynomial
    d. **cub** cubic polynomial
6. **userunits** is a short text string giving the units in which processed data is normally reported from the logger; for example **C** for Celsius, **V** for Volts, **dBar** for decibars, etc. Presently this is a factory-set field representing the fundamental units in which the channel is calibrated; support for user-selectable units is planned in the future.

7. **derived** is a flag which is either **on** or **off** to indicate whether the channel is a derived channel (**on**) or a measured channel (**off**). This is an intrinsic property of the channel **type** , and can not be modified: it is for information only.

8. **gainsavailable** reports the gain settings supported by the sensor at channel <*index*>. The settings are given as a list of numerical values in order of increasing gain, with a vertical bar character '**|**' separating the values. If the channel at <*index*> does not support multiple gain settings, the response is **none**.

The following parameters can be changed by end users, if and when appropriate.

1. **gain** reports the gain setting currently in use by the channel at <*index*>. In addition to one of the fixed values from the list reported by the **gainsavailable** option, the response may indicate **auto** for auto-ranging. In this mode the channel will select the most appropriate gain setting depending on the value of the parameter being measured. Again, if the channel does not support multiple gain settings, the response is **none**.

   The **gain** option may also be used to set the gain used. For a fixed gain setting, the value supplied must be from the list reported by the **gainsavailable** option. For auto-ranging, use the word **auto**. Although they are typically whole numbers, gains are reported in a floating point format, and may be specified as such, as long as the value appears in the list of available gains.

   > ⚠ The **gainsavailable** and **gain** parameters are only available for channel types which support sensors having variable gain, or multiple ranges. Presently these include sensors from Seapoint, and the Cyclops series from Turner Designs, which can measure turbidity, fluorescence, and various other optical properties. For a complete list refer to the Section Supported Channel Types (page 139).

2. **status** is a further basic parameter which applies to all channels. It is modifiable by end users, and allows any individual channel to be turned off or on for the duration of a deployment.

- - **on**: the channel is activated for sampling; its data will be stored in memory if appropriate, and its value will appear in streamed output if streaming is enabled. However, note that if data storage is set to Standard format (rawbin00), data is never stored for derived channels, because raw data for such channels does not exist. For the same reason, data for derived channels can not be streamed if one of the raw output formats is selected.
  - **off**: the channel is not sampled, no data will be stored in memory or streamed for this channel.

> ⚠ If you turn **off** any of the logger's channels, and intend to use RBR's Ruskin software - for example, to download data - be sure to use the latest version of Ruskin; earlier versions may not correctly handle channels which exist but have been turned off.

Examples

```
>> channel 1
<< channel 1 type = temp09, module = 1, status = on, latency = 140, readtime = 150, equation = tmp, userunits = C
```

```
>> channel 2 equation userunits
<< channel 2 equation = cub, userunits = dBar
```

```
>> channel all type
<< channel 1 type = temp09 | 2 type = pres19
```

```
>> channel 4 gainsavailable
<< channel 4 gainsavailable = 1.0|5.0|20.0|100.0
```

```
>> channel 4 gain
<< channel 4 gain = auto
```

```
>> channel 4 gain = 20
<< channel 4 gain = 20.0
```

Errors

Error E0105
Settings may not be modified while logging is in progress; reading them is permitted.

Error E0107
An argument expected by the logger was not given with the command; for example, there must always be an <index> argument.

Error E0108
This error will occur if the <index> is out of range, or if an unknown parameter is requested. Logger channels are numbered from 1 to N; zero is not valid.

Error E0111
There was a problem reading or modifying some configuration data for the specified channel; typically in response to accessing gain control information for those channels which support it. Please contact RBR Ltd for help.

Error E0505
There was a problem reading or modifying some configuration data for the specified channel; typically in response to accessing gain control information for those channels which support it. Please contact RBR Ltd for help.

Error E0501
There is a problem with the configuration of the specified channel; please contact RBR Ltd for help.

Use cases for 'status' settings:
(1) RBR*concerto* C.T.D; realtime output data (streamed or fetched) is required to show only salinity, temperature and pressure:
Ensure the derived salinity channel is **on**.

1. Turn **off** any other derived channels which are available but not required (eg. depth, sea pressure, etc).
2. Turn **off** the Conductivity channel (!!).

The logger knows it requires conductivity to compute salinity, and will still sample this 'off' channel, but will not report or store (see Note) the data.

(2) RBR*concerto* C.T.D; realtime output data (streamed or fetched) is required to show only depth.
Ensure the derived depth channel is **on**.

1. Turn **off** any other derived channels which are available but not required.
2. Turn **off** the Conductivity and Temperature channels.
3. Turn **off** the Pressure channel (!!).

The logger knows it requires pressure to compute depth, and it also knows that temperature is required for the correction of pressure, so it will still sample both of these 'off' channels, but will not report or store (see Note) the data. Conductivity is not needed, so this channel will be truly

'off'.

For storage of data in Standard format (rawbin00):

1. Only raw, uncorrected, measured channels are stored.
2. Corrected or derived channels are never stored.
3. A raw, uncorrected, measured channel will not be stored IF the channel has been turned **off**, AND no other channel depends on its data for correction.
4. A raw, uncorrected, measured channel which has been turned **off** will still be stored if any other channel depends on its data for correction.

For storage of data in EasyParse format (calbin00):

1. All channels which are **on** are stored, including any derived channels.
2. No channel which is **off** is ever stored, even if another channel depends on its data for correction. The value of the dependent channel is already computed and stored, so there is no need to also store the supporting channel unless it is **on**.

3.7.3 settings

Usage

>> settings [<parameter_name> [= <value>] …]

Security

Open for reading, Protected for update, Unsafe for update (modifications not permitted while logging).

Description

Reports or sets the values of miscellaneous settings in the logger as described below.

1. **fetchpoweroffdelay** is the delay in milliseconds between successful completion of a fetch command, and power to the front end sensors being removed by the logger. Power is left on for a short time to avoid excessive power cycling when sending repeated **fetch** commands; this parameter allows that delay to be adjusted. The default value is 8000.
2. **sensorpoweralwayson** is a flag which is either **on** or **off**. When **on**, the logger does not remove power from the front end sensors between samples. This can be useful for sensors with very long power-on stabilization times. The default setting is **off**.

3. **castdetection** is a flag which is either **on** or **off**. When **on**, the logger will detect automatically upcasts and downcasts, and will generate cast detection events in the datastream. It is advisable to ensure this option is **off** if the logger is not used as a profiler. The default setting is **off.**

4. **inputtimeout** is available only in firmware versions 1.130 or later, and specifies the value of a timeout used by the logger when receiving command input; it is used to temporarily blank other output such as streamed data, and to assist in power saving by turning off the serial communication interface if it is not needed. See the section Timeouts, Output Blanking and Power Saving (page 10) for more details. The value is specified in milliseconds; the default value for all instruments is 10000 (10 seconds). The value may be set within the range 10000 (10 seconds) to 240000 (4 minutes) inclusive; partial seconds are rounded up to the next whole second value. Instead of a numeric value, the word **default** may be used to restore the.default value of 10000.

5. **offsetfromutc** is available only in firmware versions 1.210 or later. It is intended to record the local time zone used when the logger was deployed, as an offset from Universal Coordinated Time (UTC). This can facilitate correct interpretation of the time information, even if the downloaded data file is reviewed in a different time zone. The offset is specified in hours; fractional hours are permitted to support time zones which require this, and the offset is always reported to two decimal places. When specifying a value, any simple numeric format compatible with floating point representation may be used; for example 11, +11, or 11.00 would all be accepted. Setting this parameter does *not* change the logger's time as reported by the clock (page 31) command; it is intended simply as a record of the local time zone. However, setting the date and time using the clock (page 31) command erases this setting; it must be restored if necessary. By default the parameter is In the erased state, in which case it is reported as **unknown**.

6. **speccondtempco** is available only in firmware versions 1.210 or later. It is the temperature coefficient used to correct the derived channel for specific conductivity to 25° C. Its value depends on the ionic composition of the water being monitored, and should be set to an appropriate value for best results. A typical range of values is 0.0191 to 0.0214, with the lower end suitable for KCl solutions and the upper end for NaCl solutions. When specifying a value, any simple numeric format compatible with floating point representation may be used; for example 0.02, 0.0200, or 2e-2 would all be accepted. If the parameter is never explicitly set, the default value is 0.0191, suitable for standard KCl solution.

7. **temperature**, **pressure**, **conductivity**, **atmosphere**, **density, salinity**, **avgsoundspeed** : these are default parameter values, to be used when the logger does not have a channel which measures the named parameter, but one or more cross-channel calibration equations requires it as an input.
   When specifying a value, any simple numeric format compatible with floating point representation may be used; for example 11, 11.000 or 1.10e+1 would all be accepted.

The units of these parameter values are implicit, and *must* be as shown below. If these parameter values are never explicitly set, they will have default values based on standard sea water (salinity = 35PSU, temperature = 15°C, hydrostatic pressure = 0 dbar), and one standard atmosphere for atmospheric pressure.

a. temperature in °C, default value 15.0
b. absolute pressure in dbar, default value 10.132501 (1 standard atmosphere)
c. atmospheric pressure in dbar, default value 10.132501
d. conductivity in mS/cm, default value 42.914 (not available in firmware 1.310 or later; replaced by salinity)
e. water density in g/cm3, default value 1.026021
f. salinity in PSU, default value 35 (only available in firmware 1.310 or later, replaces conductivity)
g. avgsoundspeed in m/s, default value 1506.8 (only available in firmware 1.330 or later)

Examples

```
>> settings atmosphere
<< settings atmosphere = 10.132501
```

```
>> settings density = 1.0295
<< settings density = 1.0295
```

```
>> settings sensorpoweralwayson
<< settings sensorpoweralwayson = off
```

```
>> settings castdetection
<< settings castdetection = off
```

Errors

Error E0105
Parameters may not be modified while logging is in progress.

Error E0108
The supplied argument was not valid; examples include: invalid parameter name and improperly specified value.

3.7.4 bpr

> ⚠ The following applies to loggers with channels types **bpr_00** up to **bpr_07** types (loggers manufactured previously to november 2015, or under a special request). For the newer implementation of BPR channels, please refer to derived BPR channels implementation (page 175).

Usage

>> bpr <index> [sensorsettling | integrationtime | serial | fullscale | paroscaldate | oversampling | U0 | Y1 | Y2 | Y3 | C1 | C2 | C3 | D1 | D2 | T1 | T2 | T3 | T4 | T5]

Security

Open, Unsafe for update (modifications not permitted while logging).

Description

This command is specific to loggers with one or more BPR (Bottom Pressure Recorder) channels installed. It reports or sets information regarding a BPR channel specified by <*index*>; the <*index*> argument is the same as that used for the command **channel**. A channel can be identified as a BPR channel by its reported type:

```
>> channel <index> type
<< channel <index> type = bpr_00 | bpr_01 | bpr_02 | bpr_03 | bpr_04 | bpr_05 | bpr_06 | bpr_07
```

Descriptions of the available parameters follow:

1. **sensorsettling** is the sensor settling time, in milliseconds, before any measurement is made. It is greater than 10 ms and below 65000 ms.
2. **integrationtime** is the integration time, in milliseconds, taken for one measurement. It is greater than 20 ms and below 172,800,000 ms (2 days).
3. **serial** is the Paroscientific sensor serial number.
4. **fullscale** is the range rating of the Paroscientific sensor. It is in meters.

5. **paroscaldate** is the Paroscientific calibration date and must be given in <*YYYYMMDD*> format.
6. **oversampling** is the number of readings averaged for each measurement; it should be between 1 and 128 inclusive.
7. **U0,Y1,Y2,Y3,C1,C2,C3,D1,D2,T1,T2,T3,T4,T5** are the calibration values of the sensor provided by Paroscientific, reported as floating point numbers using a format with a mantissa and exponent; for example 3.3910000e+003. When setting coefficients, any simple format compatible with floating point representation may be used; for example 11, 11.000 or 1.10e+1 would all be accepted.

Parameters may be read individually or in any combination, but setting more than 15 parameters at a time is not possible.

Examples

```
>> bpr 2
<< bpr 2 serial = 210, fullscale = 2000, U0 = 5.8, Y1 = -3954.4, Y2 = -10760.9, Y3 = .0, C1 = -29800.7, C2 = 1820.5, C3 = 106769.1, D1 = 0.0, D2 = 0.0, T1 = 30.0, T = 1.8, T3 = 65.9, T4 = 221.7, T5 = 0.0, paroscaldate = 20130115000000, integrationtime = 600, sensorsettling = 2500, oversampling = 64
```

```
>> bpr 3 sensorsettling=1000, integrationtime = 300
<< bpr 3 integrationtime = 300, sensorsettling = 1000
```

Errors

Error E0105
Parameters may not be modified while logging is in progress.

Error E0107
An argument expected by the logger was not given with the command; for example, there must always be an <index> argument, and if setting coefficients then all fields required must be supplied.

Error E0108
The supplied argument was not valid.

Error E0501
There is a problem with the configuration of the specified channel; please contact RBR Ltd for help.

3.7.5 calibration

Usage

>> calibration <index> [type] [datetime = [<YYYYMMDDhhmmss>]], [[c0 [= <value>] …] | [x0 [= <value>] …] | [n0 [= <value>] …]]

Security

Open for reading; Protected and Unsafe for update (modifications not permitted while logging).

Description

Reports or sets information regarding the most recent calibration for the channel specified by <*index*>, which is a required parameter in all cases; the <*index*> of the first channel is 1. The number and types of coefficients reported, or required when setting, will vary depending on the sensor **type**.

Some sensor types have complicated equations with many coefficients, and the equation may also use the output of one or more of the other channels in the logger for correction or compensation purposes. This is a powerful facility, but requires a lot of information; the **calibration** command helps to manage that information.

Coefficients are arranged in three groups, **c0**…, **x0**…, and there is a further group **n0**… of cross-channel reference indices. The purpose and function of each group will be described below. The groups may also be referred to by name; **c**, **x** or **n**.

If given with only the <*index*>, the command reports all information applicable to the channel. Parameters may also be requested individually, or in any combination, by name. Coefficients in each group may be requested all together by using one of the group names, **c**, **x** or **n**. Requesting an item which does not exist (eg. **c3** for a linear sensor) may result in either an error message, or a response such as **c3 = n/a**.

A special value **all** may be given for the channel <*index*>, causing the requested parameters to be reported for all channels. The output for each channel is terminated by a vertical bar character '**|**', except for the last channel which is terminated by a <**cr**><**lf**> pair as normal.

When setting parameters, there are further restrictions which must be followed:

1. Some parameters are read only; **n0…** and **type**.
2. **datetime =** <*YYYYMMDDhhmmss*> must accompany any changes to coefficient values, so that the reported date and time reflects the most recent change.
3. a single **calibration** command can set coefficient values in only one of the groups at a time; **c0**…, **x0**…, Coefficients from different groups can not be mixed in a single command when setting.

4. For firmware versions prior to 1.220, *all* coefficient values in the group must be set with one command; coefficient values cannot be set individually. This constraint attempted to ensure that each group is always consistent. For firmware versions 1.220 or later, this constraint has been dropped for convenience; any number of coefficients in a single group may be changed with one command.

Descriptions of the individual parameters are given below.

1. **type** is a read-only parameter created during factory configuration; it can not be modified. It describes the channel type and determines the calibration equation used, and hence the number and type of coefficients required.
2. **datetime** is reported and set using a <*YYYYMMDDhhmmss*> format. It is the date and time of the most recent calibration change for the channel, and is a required parameter when setting any of the calibration coefficients.
3. **c0, c1**… are the primary coefficient values, reported as floating point numbers using a format with a mantissa and exponent; for example 3.3910000e+003. When setting coefficients, any simple format compatible with floating point representation may be used; for example 11, 11.000 or 1.10e+1 would all be accepted.
   These coefficients apply to a 'core' equation which yields a basic value for the parameter. In many cases this is all that is needed, and the **x** and **n** groups are not required. The exact function of each coefficient depends on the equation used.
4. **x0, x1**… are required and reported for only some equation types, namely those which employ cross-channel compensation or correction of the primary value using one or more inputs from other channels in the logger. **x0, x1**… are also coefficient values which follow the same rules as the **c** group. The exact function of each coefficient depends on the equation used.
5. **n0, n1**… apply only to some equation types, those using cross-channel compensation or correction. They are only ever reported; they are set at the factory and can not be changed. They are not coefficients, but (in general) the indices of other logger channels whose data are also inputs to the equation for channel <*index*>. This permits output data to depend on more than one channel; for example, to be corrected for temperature dependencies. The values of **n0, n1,** etc. are simple integer numbers, remembering that the index of the first channel is 1; zero is not valid.

Equations which use the **x0, x1**… coefficients will require at least one '**n**' index. The logger may also have 'derived parameter' channels, which have no measurement channel of their own, but an output value which is computed from other measured channels: a good example would be salinity, which is a function of conductivity, temperature and pressure. In such cases **n0, n1, n2** are required to tell the logger which input channels to use.

There is one special case when the value of an '**n**' index may be the text field "**value**". This can be set only at the factory, and applies when an equation requires a correction term using a parameter which the logger does not measure. In this case the default parameter set by the command **settings** will be used.

Please refer to the section Calibration Equations and Cross-channel Dependencies (page 148) for a complete list of the equations which the logger uses, and for further discussion of cross channel dependencies,.

Examples

```
>> calibration 1
<< calibration 1 type = volt00, datetime = 20110518175005, c0 = 9.9876543e+000, c1 = 7.5642301e+000
```

```
>> calibration 1 datetime = 20120503134201, c0 = 9.9873456, c1 = 7.564
<< calibration 1 type = volt00, datetime = 20120503134201, c0 = 9.9873456e+000, c1 = 7.5640000e+000
```

Errors

Error E0103
"**permit calibration**" must immediately precede the command if setting coefficients.

Error E0105
Coefficients may not be modified while logging is in progress.

Error E0107
An argument expected by the logger was not given with the command; for example, there must always be an <index> argument, and if setting coefficients then all fields required must be supplied.

Error E0108

The supplied argument was not valid; examples include:

- *<index>* out of range; channels are numbered from 1 to N; zero is not valid.
- improperly formatted or invalid *<datetime>* argument.
- invalid coefficient name.
- improperly specified value for a coefficient.

Error E0501
There is a problem with the configuration of the specified channel; please contact RBR Ltd for help.

Error E0505
Indicates a serious fault with the logger; please contact RBR Ltd for help.

3.7.6 sensor

Usage

>> sensor <index> [<parameter_name> [= <parameter_value>], ...]

Security

Open, Unsafe (modifications not permitted while logging).

Description

Returns information about the sensor attached to the channel at the specified <*index*>: the first channel has an <*index*> of 1.

This command is available only with firmware versions 1.220 or later. Its purpose is to manage miscellaneous information relating to the *sensor* associated with a given logger channel, as opposed to any property of the channel itself; the distinction is rather fine. In general the **sensor** command is used for information which belongs with a particular sensor, but which the logger does not need to know; a good example would be the sensor's serial number. For information common to all sensors of this type which the logger *does* need to know, the channel (page 78) command is used; a good example would be the **latency**, or power-on settling delay.

In principle any channel type could make use of the **sensor** command; in practice the channels which use it and the parameters which are supported are defined by the instrument's Factory configuration. End users may change the values of existing parameters, but they can not add new parameters or add the capability to channels which do not already have it. Attempting to use the sensor command with a channel not configured to support it will provoke an error message, as detailed below. Channel types and parameters which may currently be configured in an instrument include the following; new combinations may be added in future.

Channel type	Parameter name	Firmware version	Comment
bpr_08	**serial**	1.220	Sensor serial number
bpr_09	**serial**	1.220	Sensor serial number

Examples

```
>> sensor 3
<< sensor 3 serial = 129837
```

```
>> sensor 3 serial
<< sensor 3 serial = 129837
```

```
>> sensor 3 serial = 119945
<< sensor 3 serial = 119945
```

Errors

Error E0105
Parameters may not be modified while logging is in progress; reading them is permitted.

Error E0107
An argument expected by the logger was not given with the command; for example, there must always be an <*index*> argument.

Error E0108
This error will occur if the <*index*> is out of range, or if an unknown parameter is requested. Logger channels are numbered from 1 to N; zero is not valid.

Error E0109
The channel selected by <*index*> is not configured to support auxiliary information via the **sensor** command.

Error E0111
There was a problem reading or modifying some data for the specified parameter and/or channel. Please contact RBR Ltd for help.

Error E0505
There is a serious problem with the logger's configuration. Please contact RBR Ltd for help.

Error E0501
There is a problem with the configuration of the specified channel; please contact RBR Ltd for help.

3.8 Communications

3.8.1 link

Usage

>> link

Security

Open.

Description

Returns the name of the communications link over which the command was received, allowing the host to determine whether a genuine serial link is in use, or the CDC serial profile of a USB connection.

Examples

```
>> link
<< link = usb
```

```
>> link
<< link = serial
```

Errors

None.

3.8.2 serial

Usage

>> serial [baudrate [= <baudrate>]], [mode [= <mode>]]

Security

Open.

Description

This command can be used to either report or set the parameters which apply to the serial link. The command can be issued over either the USB or serial links, but care must obviously be taken if the serial link is used to change its own operating parameters. In this case, new settings are acknowledged while the old parameters are still in force, then the changes are applied. The next command sent must use the new configuration of the link if the logger is to recognize it. The individual parameters are described below.

baudrate: the following baud rates are supported: 115200, 19200, 9600, 4800, 2400, 1200. The default shipped from the factory is 115200Bd.

mode: this parameter allows the electrical interface standard used for the serial link to be changed, the available choices being listed below. Different modes typically require differences in hardware, so changing modes may not always be appropriate. The most common mode is RS-232, and this is the default setting typically shipped from the factory. If an instrument has been built to use one of the other interfaces, the mode will be correctly set when the instrument is shipped.

- **rs232**: This is the legacy standard used by default on most equipment with serial ports, referred to as RS-232, EIA-232, TIA-232, or variations on one of these depending on the revision, but for most practical purposes they are interchangeable. The logger's implementation of RS-232 is always full duplex, with no hardware flow control lines required: transmit, receive and ground are the three connections needed.
- **rs485f**: This is the full duplex version of the RS-485 standard (also EIA-485, TIA-485, etc), which permits higher speeds and/or longer distances than RS-232. A five-conductor cable is required; two lines each for both receive and transmit, plus a ground connection. In most cases a simple cable will work, but at extreme speeds and distances, the transmit and receive line pairs may require impedance matching termination components. The logger does not include these, as they will be specific to each individual installation.
- **rs485h**: This mode is planned, but not yet supported on any loggers. It is the half duplex version of RS-485, which requires only three connections: ground plus a data line pair which is used for both receive and transmit.
- **uart**: This offers logic level (0-3.3V swing) serial interface to the UART on the logger’s serial port. The “idle” state of the line, i.e. the state of the serial transmit line during the time before and after transmission of data bytes, is high (3.3V). This may be a useful option for OEM integrators, typically over short distances to another piece of equipment, where the communication link is not exposed to the outside world. In this mode, it is worth noting that the serial receiver interface on the logger has a (nominal) 5K pulldown

resistor to 0V in the circuit at all times. As such, in order to minimize current consumption while there is no serial activity, it is recommended that the serial transmit signal coming from the circuit that the logger is interfaced to is either tristated off (high impedance) or held at a logic low (0V).

- **uart_idlelow**: the same as **uart**, but with inverted logic levels, so that the "idle" state is low (0V). This may be thought of as the same logic states as RS232, except that it utilizes 0-3.3V logic levels.

Examples

```
>> serial
<< serial baudrate = 19200

>> serial baudrate = 115200
<< serial baudrate = 115200
```

```
>> serial mode
<< serial mode = rs232

>> serial mode = rs485f
<< serial mode = rs485f
```

Errors

Error E0108
The supplied argument was not a recognized parameter name, baud rate value, or mode setting.

Error E0104
Half duplex RS-485 mode is planned, but not yet available.

Error E0114
To avoid permanently disrupting a communications link, changing any of the serial link's operating parameters is not permitted if a WiFi module is in use, even if the logger would otherwise support these options.

3.8.3 sleep

Usage

>> sleep

Security

Open.

Description

Immediately shuts down communications and implements any power saving measures which are possible, over-riding the 10-second timeout which normally invokes these actions (see Section Timeouts, Output Blanking and Power Saving (page 10)). Power saving measures typically include:

- any interface circuitry used for a Serial link,
- sensor channels activated *only* for the purpose of satisfying a fetch (page 104) command.

Any scheduled sampling activity is not affected. The **sleep** command does not attempt to power down a USB link, because there is always enough power available via USB to run the logger's basic functions; sensor channels used for a **fetch** command will still be shut down.

The command generates no error messages, and always succeeds silently; there is no prompt or confirmation message in response to the command.

3.8.4 wifi

Usage

>> wifi [timeout [= <timeout>]], [commandtimeout [= <commandtimeout>]]

Security

Open.

Description

This command allows the user to define two timeouts for the wifi unit. The parameters currently supported are:

1. **timeout** corresponds to the timeout in seconds the wifi module will use until the first valid command is received since the wifi module has been powered up. If the timeout expires the wifi module is powered down. The value is between 5 and 600 seconds.
2. **commandtimeout** is the timeout used upon the reception of a valid command by the logger. If the timeout expires, the wifi module is powered down. The value is between 5 and 600 seconds.

Examples

```
>> wifi timeout = 120
<< wifi timeout = 120 commandtimeout = 60
>> wifi
<< wifi timeout = 120 commandtimeout = 60
```

Errors

Error E0108
The command was given with an argument which is unrecognized or has an invalid value; for example "wifi timeout = 4".

3.9 Other Information

3.9.1 id

Usage

>> id [model | version | serial | fwtype | all]

Security

Open.

Description

This is a read-only command which identifies the logger, reporting the model name, the version of firmware in the CPU, the unit serial number, and a firmware type. The serial number is always reported using six digits, padded with leading zeroes if necessary.

The **fwtype** parameter is used to identify different products among the range of RBR's dataloggers. The **fwtype** for all RBR*virtuoso*, RBR*duo*, RBR*concerto*, and RBR*maestro* models early 2015 and onward is 103.

⚠ Versions of RBR*virtuoso*, RBR*duo*, RBR*concerto*, and RBR*maestro* models prior to 2015 report a **fwtype** of 100.

Examples

```
>> id serial
<< id serial = 050032
```

```
>> id
<< id model = RBRconcerto, version = 1.000, serial = 050032, fwtype = 103
```

For firmware versions 1.300 or higher, the response may include an indication that simulated data is enabled as shown below. Refer to the command simulation (page 67) for more details.

```
>> id
<< id mode = SIMULATION, model = RBRconcerto, version = 1.300, serial = 012345,
fwtype = 103
```

Errors

None.

3.9.2 powerstatus

Usage

>> powerstatus [source | int | ext | capacity | powersupply | all]

Security

Open.

Description

Reports parameters relating to the logger's power sources as follows:

1. **source** is one of the following names:
    a. **usb** the logger is drawing power from the USB connection.
    b. **int** the logger is drawing power from its Internal battery.
    c. **ext** the logger is drawing power from an External power source.
2. **int** is for reporting the measured voltage of the logger's Internal battery.
3. **ext** is for reporting the measured voltage of any External power source attached.

4. **capacity** is the nominal power capacity of the Internal battery in Watt-hours (Wh), reported as a floating point number with three decimal places, for example: 24.000. The units are not reported.
5. **powersupply** is a compact description of the logger's configured options for power.

The **powersupply** parameter reports the nominal voltage rating (not the actual measured voltage) for each of the internal battery and, if available and used, the external source:

- **12i12e**: nominal 12V internal battery, external source must be compatible with a nominal 12V.
- **12i24e**: nominal 12V internal battery, external source must be compatible with a nominal 24V.
- **24i12e**: nominal 24V internal battery, external source must be compatible with a nominal 12V.
- **24i24e**: nominal 24V internal battery, external source must be compatible with a nominal 24V.

The recommended minimum and maximum values for external supplies are:

- 12V: 9V to 15V
- 24V: 18V to 28V

Exceeding the maximum values may result in damage to the logger and/or any external sensors attached. Failing to meet the minimum may result in unpredictable behaviour and/or inaccurate data.

Examples

```
>> powerstatus
<< powerstatus source = usb, int = 12.40, ext = 0.00, capacity = 24.000
```

```
>> powerstatus int
<< powerstatus int = 12.39
```

Errors

Error E0108

The supplied argument was not a recognized parameter name.

Error E0111

A requested parameter could not be measured because of an A/D converter failure.

3.9.3 errorlog

Usage

>> errorlog [clear]

Security

Open for reading, Protected to clear.

Description

The error log contains run-time errors detected in the logger's internal operation which can not easily be reported directly to the end user.

> Such errors are rarely encountered, but if they are, the error log provides valuable diagnostic information which should be noted down and reported to RBR Ltd. The information can not be interpreted by end users.

Such run-time errors are normally recorded in the logger's data memory, where they can be identified in the downloaded data and properly interpreted by the host software. The errorlog command does not show these errors. There are two situations in which the above mechanism can not be used:

1. When the error indicates a fault with the memory itself, and
2. Before logging has been enabled, so the data memory is not in use.

In these cases, a record of only the most recent error is kept, and this is what the **errorlog** command reports. An instrument can not be enabled for logging unless **errorlog** reports **clear**. After noting the error details, clear the log using the optional **clear** parameter to the command. Clearing the error log is a protected operation; the command **permit errorlog** must be issued immediately beforehand.

Examples

```
>> errorlog
<< errorlog = clear
```

```
>> errorlog
<< errorlog = 20110928165600, 0x00834C, 0x12345678
```

```
>> permit errorlog
<< permit = errorlog
>> errorlog clear
<< errorlog = clear
```

Errors

Error E0103
"**permit errorlog**" must immediately precede the command if clearing the error log.

Error E0108
The supplied argument was not "clear", the only recognized parameter.

3.9.4 help

Usage

>> help [<command-name>]

Security

Open.

Description

The help command, without arguments, generates a list of all known commands along with possible parameters and a short description of functionality. If a single valid command name is passed to the help command, only the description for that command is returned.

Examples

```
>> help id
<< id [model|version|serial|fwtype]: report unit identification
```

Errors

Error E0102
Help was requested for an unknown command.

3.9.5 hwrev

Usage

>> hwrev [pcb | cpu | bsl]

Security

Open.

Description

Reports various pieces of information about the revision status of the logger's main circuit card. Not usually of interest except for diagnostic purposes, or to determine whether hardware-dependent features may or may not be available in advance of trying to use them.

The elements of the response are as follows:

- <pcb> is a letter such as 'C', 'F', etc., which represents the revision level of the main Printed Circuit Board inside the logger.
- <cpu> is a number and letter, identifying the type and silicon revision of the CPU chip used on the main PCB.
- <bsl> is a letter giving the version of a firmware component used to reprogram the CPU chip in-situ.

Examples

```
>> hwrev
<< hwrev pcb = F, cpu = 5529E, bsl = C
```

Errors

None.

3.9.6 altitude

Usage

>> altitude [= <value>]

Security

Open.

Description

Available only if the logger is configured to support the **wave** sampling mode.

Reports or sets the 'altitude', or height above the sea bed, at which the logger is deployed. This is a user-entered parameter which is required by host software to calculate statistics and parameters for wave analysis: it is not used internally by the logger, and if wave analysis is not required the parameter can be ignored.

<*value*> is specified in floating point form, for example 10.3, and is reported with two decimal places, for example 10.30. If the parameter is not set, the default value is 0.00. There is no error checking on the value entered, and the units are not specified, although RBR Ltd's host software Ruskin expects the value to be in metres.

Examples

```
>> altitude
<< altitude = 20.50
```

```
>> altitude = 5
<< altitude = 5.00
```

Errors

Error E0102
The logger is not configured to support the **wave** sampling mode.

Error E0108
The argument given was not a valid number.

3.10 Data sample

3.10.1 fetch

Usage

>> fetch [sleepafter = true|false]

Security

Open.

Description

Requests an 'on-demand' sample set from the logger. If a recent scheduled sample set is available, those values may be returned to satisfy the **fetch** request. 'Recent' in this context currently means less than 500ms old. Otherwise, a sample set is explicitly acquired for the benefit of the **fetch**. A sample set acquired only for **fetch** is never stored in memory.

The logger simply responds with the <*sample-data*>; depending on the configured latency (or power-on settling delay) for the attached sensors, there may be a noticeable delay before the <*sample-data*> appears. Refer to the channels (page 77) command for further discussion of latency. The output format of the <*sample-data*> is determined by the outputformat (page 52) command.

Subsequent **fetch** commands may return data more quickly; after an initial **fetch** command, the sensors usually remain powered up in anticipation of another request. This behaviour is protected by an 8-second (default) timeout, after which the sensors are turned off again. Refer to the settings (page 83) command for information on changing the default sensor power-off timeout.

This behaviour may also be modified by using the **sleepafter = true** option, which causes the logger to power down when it has finished reporting the <*sample-data*>. This is equivalent to issuing two separate commands, **fetch** then sleep (page 95), except that after completing the **fetch** the logger goes to sleep silently; there is no "Ready" prompt following the <*sample-data*>, just as there is no "Ready" prompt following a sleep (page 95) command. Note that the entire logger powers down if possible, not just the sensors; however if any normally scheduled samples are required immediately after the **fetch**, the power down action will be delayed until after those samples are complete.

Specifying **sleepafter = false** does not provoke an error message from the logger, but it has no effect either; the **fetch** command behaves as if no option had been specified.

Examples

```
>> fetch
<< 2012-10-21 11:50:49.000, 18.1745 C, 12.7052 dBar
```

Errors

Error E0108
An unrecognized argument was given with the command.

Error E0107
If the **sleepafter** option is used, a value of either **true** or **false** must also be supplied.

Error E0410
Indicates that the logger has no channels activated for sampling.

Error E0111
Indicates a serious fault with the logger; please contact RBR Ltd for help.

3.11 Security and Interaction

3.11.1 permit

Usage

>> permit <command-name>

Security

Open.

Description

Permits a protected command to be executed immediately after this one; receipt of anything else removes the permission again. Any other constraints on executing a particular command will still apply.

It is not an error to '**permit**' a command which does not need it, merely unnecessary.

Examples

```
>> permit memclear
```

```
<< permit = memclear
>> memclear
<< memclear used = 0
```

Successfully clears the data memory.

```
>> memclear
<< E0103 protected command, use 'permit memclear'
```

Fails because memclear is a protected command.

```
>> permit memclear
>> id
>> memclear
<< E0103 protected command, use 'permit memclear'
```

Fails because **permit** must *immediately* precede the protected command.

Errors

Error E0107
No <command-name> argument was given.

Error E0108
The <command-name> argument given is not a recognized command.

3.11.2 prompt

Usage

>> prompt [state [= on | off]]

Security

Open.

Description

Returns the state of the "**<cr><lf>Ready:** " prompt, which is normally sent by the logger in response to almost any command after any other output generated by the command is complete. If '**on**', the prompt is sent; if '**off**' the prompt is suppressed.

A change of the on/off state takes place immediately, so for example there will be no prompt following the command which turns it off. Turning the prompt off is not normally recommended, unless there is a very good reason for doing so. For example, if it is interfering with the parsing of responses by an automated system, it may be necessary to suppress it.

Examples

```
>> prompt
<< prompt state = on
```

```
>> prompt state = off
<< prompt state = off
```

Errors

Error E0108
An unrecognized argument was given.

3.11.3 confirmation

Usage

>> confirmation [state [= on | off]]

Security

Open.

Description

Returns the state of the logger's confirmation responses, normally sent after a parameter has been modified if the state is **on**. If the state is **off**, *successful* parameter modifications occur without confirmation messages.

A change of the on/off state takes place immediately, so for example there will be no confirmation of the command which turns it **off**.

There are several situations in which the suppression does not occur even if the state is **off,** here are some points to note:

1. Requests to simply report a parameter always generate output.
2. Error messages resulting from a *failed* attempt to set a parameter are always sent.
3. Some 'action' commands such as **enable** always generate a confirmation message.
4. The "**<cr><lf>Ready:** " prompt is controlled separately by the **prompt** command.

Turning confirmation off is not normally recommended, unless there is a very good reason for doing so.

Examples

```
>> confirmation
<< confirmation state = on
>> confirmation state = off
<<
```

Confirmation of a parameter change is immediately suppressed.
The following example commands are all with confirmation state = off:

```
>> starttime
<< starttime = 20110507130000
```

A request for information always provokes a response.

```
>> starttime = 120601120000
<< E0108 invalid argument to command: '120601120000'
```

A failed attempt to set a parameter still provokes a response.

```
>> starttime = 20120601120000
<<
```

Response suppressed if parameter is successfully changed.

Errors

Error E0108
An unrecognized argument was given.

3.11.4 reboot

Usage

>> reboot [<milliseconds-delay>]

Security

Protected

Description

This command executes a logger CPU reset. The reset will apply only to the CPU itself and any hardware directly under its control; there is no guarantee that every component in the logger system will be reset in the same way that cycling power to the logger would achieve.

The optional delay argument in milliseconds is useful when using the command over a USB-CDC communication link. When the logger CPU resets, any USB-CDC link between it and the host will be torn down and then re-established, meaning that the virtual serial port associated with the CDC profile temporarily disappears for a brief time. Most communications software is unable to cope with such an event, so providing some time to disconnect the software from the logger before the port disappears allows the operation to be performed gracefully.

This does not apply to a true Serial link, so there are no side effects if the logger is reset without specifying a delay. The link (page 93) command can be used to verify the type of communications link if there is any doubt.

Except in the case of the error message reported if the permit (page 105) mechanism is not used, there is no response to the command: once the reset occurs the logger no longer has any memory of receiving the command, so it can not respond.

Examples

```
>> permit reboot
<< permit = reboot
>> reboot
```

Successfully resets the logger CPU.

```
>> reboot
<< E0103 protected command, use 'permit reboot'
```

Fails because **reboot** is a protected command.

```
>> permit reboot
<< permit = reboot
>> reboot 5000
```

Successfully resets the logger CPU after a delay of five seconds.

Errors

Error E0103
permit reboot must immediately precede the command to reset the logger.

4 Parameter Modification

All updated parameters are held temporarily in a RAM buffer, and read back from there if interrogated. The data is permanently stored under the following conditions:

1. timeout protection, 10 seconds after the last parameter modification.
2. successfully enabling the logger to sample.
3. executing the **sleep** command.

If none of these conditions are met (removal of power before timeout, for instance), parameter values may not be those expected. This could apply if, for example, a logger is programmed via USB, without internal batteries installed and relying on the USB for power. If the USB link is unplugged before the logger has a chance to save any changes made, they will be lost.

5 Format of Stored Data

5.1 Overview

There are three major types of deployment information stored in a logger's memory:

1. a deployment header, which contains meta-information about the logger and the deployment parameters.
2. sample data, comprising sets of measured values from all active channels in the logger.
3. events, which are records of non-sample incidents used to aid interpretation, or for diagnostics.

Below is a brief overview of the available storage formats used to record all this information: detailed descriptions are presented in later sections. To determine which storage formats are available, use the **memformat support** command. If the logger does not recognize this command, then Standard format is the only one supported.

5.1.1 Standard format

All loggers support a data format known colloquially as 'Standard', and formally as 'rawbin00'. In Standard format, the three major types of deployment information are stored in a single dataset. This has the benefit that retrieving the data ensures that all information relevant to the deployment is present, producing a complete, self-contained deployment record in a single download operation.

The organization of the stored information within the dataset is as follows:

1. the deployment header is always stored at the beginning, then
2. following the header, sample data is stored as it accumulates, while
3. events are stored as they occur, embedded in the stream of sample data.

A further benefit to this approach is that because items are stored chronologically, much of the date/time information - in particular for sample sets - is implicit, and no memory is consumed by additional timestamp information. Disadvantages are that the data needs to be parsed carefully to interpret it properly, and that there is no 'local context' for isolated snippets of data: in general the entire dataset is needed to ensure all timing information is correctly decoded.

The format used must be set before a deployment is started by the memformat (page 75) **newtype** command. When downloading data, use the memformat (page 75) **type** command to determine the format of the data which is currently in memory. Downloading data is performed using the read (page 72) command on dataset-1.

5.1.2 EasyParse format

Loggers may also support a data format known colloquially as 'EasyParse', and formally as 'calbin00'. In EasyParse format, each of the three major types of deployment information is stored in its own separate dataset. This has the benefit that parsing two of the three types (sample data and events) can be greatly simplified, with the penalty that a single download operation will not produce a complete, self-contained deployment record: if this is a requirement, it becomes the responsibility of the user and/or host software.

The assigment of datasets to the stored information is as follows:

1. the deployment header is in dataset 2,
2. the sample data is in dataset 1, and
3. events are in dataset 0.

The read (page 72) command acting on each of these three datasets as required is used to download the information: use the memformat (page 75) **type** command to determine the format of the data currently in memory. Host applications using the EasyParse format may elect not to download the deployment header at all; if only a small subset of the information it contains is needed, it may be easier to determine that using other logger commands.

5.2 EasyParse "calbin00" format

5.2.1 Sample data EasyParse format

In EasyParse format (calbin00), dataset-1 contains *only* sample data, comprising sample sets recorded in chronological order. The format of an individual sample set is also quite different from that of Standard format. Every sample set includes a timestamp, and values are already converted to the physical values of the required parameters according to the logger's calibration, with correction or compensation already applied as necessary. They are also stored in a different numeric format, with error codes still accommodated when required. Refer to the following sections for more details.

The logger can also store derived channels such as depth or salinity, which have no corresponding 'raw' value in the Standard data format.

These characteristics make the data easier for host software to interpret:

1. All stored items are sample sets, with the size fixed for a given logger deployment; 8 bytes for a timestamp, plus 4 bytes for each channel stored.
2. Host software does not need to do any calculation or need to know the logger's calibration data; final values are stored directly by the logger.

3. Derived channels are already included.

Sample timing

Sample sets are individually time-stamped; while this consumes memory, it does also mean that timing information is always available for snippets of downloaded data.

Each timestamp is a 64-bit (8-byte) unsigned integer, representing the number of milliseconds elapsed since 1970-Jan-01 00:00:00. This is a format commonly used by Unix-based computer systems, in which leap years are correctly accounted for, but each day is assumed to contain exactly 86400 seconds; there is no allowance for leap seconds or other obscure adjustments. The logger takes no account of time zones or daylight savings adjustments.

If a sample set is the result of an average or a bin, the timestamp reflects the timestamp of the first measurement of the average or bin.

Normal reading values

Each individual reading of the sample data is stored as a 32-bit (4-byte) floating point number in IEEE-754 single precision format. Both measured and derived channels are included in the sample set, with channels ordered as expected according to the results of the channel (page 78) command. The values are the final computed output for each channel, including all necessary corrections and cross compensations. If an error occurs on an individual channel in a sample set, that channel will be reported as an IEEE-754 'NaN' (Not a Number): see below for more details.

Error Codes

Under some conditions an error may occur on one channel while data from the other channels is perfectly acceptable. Rather than generating a time-stamped event if this happens, the individual reading is replaced by an error code.

In EasyParse format, an error code is stored as an IEEE-754 'NaN' (Not a Number), which is compatible with the floating point format of the sample readings, without being a valid value. An error code indicates a problem with that particular reading from the channel in question; other readings in the same sample set may be fine, as may other readings from the same channel in different sample sets.

IEEE-754 provides for multiple NaN values, and this feature is used to encode the nature of the error, although it is acknowledged that host software taking advantage of the simplicity of the EasyParse format will probably not delve into this level of diagnostic detail. The general format is (0xFF800000 + <*EC*>), where 0xFF800000 is the base value of an IEEE-754 NaN, and <*EC*> is the error code. The values of <*EC*> correspond, where possible and appropriate, to the error numbers also used in Standard data storage format.

Error #	Hex Code	Description
-	0xFF800001	internal computation failure (eg. divide-by-zero)
-	0xFF800002	unable to compute value, channel not calibrated
0	0xFF810000	generic, unknown or unexpected error
1	0xFF810001	EOC bit unexpectedly set in ADC output
2	0xFF810002	DMY bit unexpectedly set in ADC output
3	0xFF810003	internal addressing error
4	0xFF810004	too much data for internal transfer
5	0xFF810005	access to internal bus denied
6	0xFF810006	timeout sending internal command
7	0xFF810007	timeout receiving internal response
8	0xFF810008	generic failure to interpret response
9	0xFF810009	no sample was started
10	0xFF81000A	sample acquisition still in progress
11	0xFF81000B	sample process failed
12	0xFF81000C	no valid samples to average
13	0xFF81000D	internal response unexpectedly short
14	0xFF81000E	supporting channel value not valid, or unknown equation
15	0xFF81000F	(reserved)

Error #	Hex Code	Description
16	0xFF810010	channel value is outside reasonable range
17	0xFF810011	channel value is below minimum measurable limit
18	0xFF810012	channel value is above maximum measurable limit

5.2.2 EasyParse format events markers

Events are records of non-sample incidents, and can be used to aid interpretation of the deployment data, or for diagnostic purposes. EasyParse events are stored in dataset-0, separate from the sample data in dataset-1.

In EasyParse format, all events have the same structure; this helps to make the stream of events easier to parse because they all have the same, fixed size: 16-bytes. Compared to Standard events, the format of the timestamp is different, and all events have an extended data field, or payload. If the payload is not applicable to a particular event type, its content is not defined, and no attempt to interpret it should be made.

For consistency, the Type Codes are the same as used for Basic (F7) and Extended (F5) events in Standard data storage format, and all are listed below for completeness. However, note that some type codes should never be encountered in EasyParse events; for example, 0x01 used for time synchronization is completely redundant, as all samples and events have their own timestamp.

Bytes 0,1	16-bit byte-swapped CCITT CRC of Bytes 2..15
Byte 2	Type Code
Byte 3	0xF4 marker byte
Bytes 4..11	64-bit unsigned number of milliseconds since 1970-01-01 00:00:00
Bytes 12..15	32-bit (4-byte) payload dependent on 'T ype Code'

The event payloads depend on the event Type Code:

1. Most event types have no payload and the content is not defined; it should be ignored.
2. A separate time-stamp field for milliseconds is redundant and never used.

3. For regime bin events (0x20), the payload is the number of reading values in the reported average.
4. For cast events, the payload is the address of the sample ***in dataset-1***, where the actual sample data is stored.

> In the case of event 0x23 (End of profiling cast), the sample address given is that of the next sample, ie. the first sample which is not in the cast.

Hex Code	Description	Payload
0x00	Unknown or unrecognized events	undefined
0x01	Time synchronization marker	undefined
0x02	**stop** command received	undefined
0x03	Run-time error encountered	undefined
0x04	CPU reset detected	undefined
0x05	One or more parameters recovered after reset	undefined
0x06	Restart failed : Real Time Clock/Calendar contents not valid	undefined
0x07	Restart failed : logger status not valid	undefined
0x08	Restart failed : primary schedule parameters not be recovered.	undefined
0x09	Unable to load alarm time for next sample	undefined
0x0A	Sampling restarted after resetting Real Time Clock (RTC)	undefined
0x0B	Parameters recovered, sampling restarted after resetting RTC.	undefined

Hex Code	Description	Payload
0x0C	Sampling stopped, end time reached	undefined
0x0D	Start of a recorded burst	undefined
0x0E	Start of a wave burst	undefined
0x0F	(reserved)	undefined
0x10	Streaming now OFF for both ports	undefined
0x11	Streaming ON for USB, OFF for serial	undefined
0x12	Streaming OFF for USB, ON for serial	undefined
0x13	Streaming now ON for both ports	undefined
0x14	Sampling started, threshold condition satisfied	undefined
0x15	Sampling paused, threshold condition not met	undefined
0x16	Power source switched to internal battery	undefined
0x17	Power source switched to external battery	undefined
0x18	Twist activation started sampling	undefined
0x19	Twist activation paused sampling	undefined
0x1A	WiFi module detected and activated	undefined
0x1B	WiFi module de-activated; removed or activity timeout	undefined
0x1C	Regimes enabled, but not yet in a regime	undefined
0x1D	Entered regime 1	undefined

Hex Code	Description	Payload
0x1E	Entered regime 2	undefined
0x1F	Entered regime 3	undefined
0x20	Start of regime bin	number of readings in average; see note below
0x21	Begin profiling 'up' cast	address of sample in dataset-1
0x22	Begin profiling 'down' cast	address of sample in dataset-1
0x23	End of profiling cast	address of first sample not in cast
0x24	Battery failed, schedule finished.	undefined
0x25	Directional dependent sampling, beginning of fast sampling mode	undefined
0x26	Directional dependent sampling, beginning of slow sampling mode	undefined

Firmware versions 1.100 or later may also support a derived data channel, type **cnt_00** , which contains this value from event 0x20 when in the regimes sampling mode. The benefit of turning it on when storing data in EasyParse format is that the value is then available in the main sample data in dataset-1; the dataset containing these events does not have to be retrieved. Refer to the Section "Integrating with a profiling float (page 18)" for further details.

5.3 Standard "rawbin00" format

5.3.1 Deployment Header

The deployment header is the first thing stored in memory by the data logger; it is written at the time of a successful **enable** command, and contains meta-information about the deployment and about the logger.

The details of the header contents do not depend on the data storage format used; only the location of the header is affected:

1. in Standard format (rawbin00), the header is stored at the beginning of dataset-1, prior to any sample data or events.
2. in EasyParse format (calbin00), the header is stored in dataset-2 by itself. This header is **not** necessary for complete EasyParse downloads.

The section which follows describes only the current version; in general, newer versions contain only additions to previous versions. A brief history of the versions is given below.

Version	Brief description of changes	Main related command
1.014	Added support for directional dependent sampling.	
1.013	Added settings avgsoundspeed.	settings (page 83)
1.012	The front-end specific section **sensor** command information reviewed. Added settings salinity.	sensor (page 91)settings (page 83)
1.011	Support for simulation mode.	simulation (page 67)
1.010	The front-end specific section may now contain information from the **sensor** command.	sensor (page 91)
1.009	Added clock offset from UTC (Universal Coordinated Time).	settings (page 83)
1.008	Added AUX1 control output parameters. Added WiFi operating parameters.	streamserial (page 57) wifi (page 96)

Version	Brief description of changes	Main related command
1.007	Added Serial communications mode. Added firmware type.	serial (page 93) id (page 97)
1.006	Added regimes information.	regimes (page 44), regime (page 46)
1.005	Added default parameters values.	settings (page 83)
1.004	Channel calibrations now support more than eight coefficients in total.	calibration (page 88)
1.003	Header size is now variable to accommodate front-end-specific information.	n/a
1.002	Added power-off delay for **fetch** command.	settings (page 83)
1.001	Added thresholding information.	thresholding (page 40)
1.000	Original release of this overall format.	n/a

Version 1.014

Headers of version 1.014 are generated by firmware version 1.360 or later.

This is a series of sections, each of which describe some aspect of the instrument configuration or deployment settings. Sections should be in ascending order by 'type' value. The 'metadata' section describes the data format itself.

Each section consists of the following:

1. Type
2. Length
3. Data structure

Section types are 1 byte values which identify the type of information found inside a section. The section types defined are as follows:

1. 0x01 Metadata
2. 0x02 Deployment

3. 0x03 Channel

Section lengths are 2 byte values which indicate the length of the data contained within the section including the section type and length.

A section's data structure format is hardcoded depending on the section type and the header version number. Unused bytes following the last entry are set to 0xFF, except for the final two bytes. These contain a 16-bit CRC using the CCITT polynomial f(x) = x^16 + x^12 + x^5 + 1, with bytes fed into the generator lsb first.

Metadata section

Type : 0x01 (1 byte)

Length : 9 (2 bytes)

Data structure content :

Version
1014 (4 bytes) This is the version of the header format

Total header length
1024 (2 bytes) This length includes everything associated with the header including the CRC at the end.

Deployment section

Type : 0x02 (1 byte)

Length : 503 (2 bytes)

Data structure : Unless stated otherwise, all parameters are 4 byte integer values.* Following the parameters, unused bytes are padded with value 0xFF.

Firmware version
Given as an integer; for example version 10.570 would be given as 10570.

Logger Serial Number
As reported by the **id** command.

Logger date/time
This is the date and time at which the enable command was successfully executed by the logger and the Header stored in memory. The value is the number of seconds elapsed since 2000-Jan-01 00:00:00.

Schedule start time
Programmed by the **starttime** command. The value is the number of seconds elapsed since 2000-Jan-01 00:00:00.

Schedule end time
Programmed by the **endtime** command. The value is the number of seconds elapsed since 2000-Jan-01 00:00:00.

Measurement interval
Programmed by the **sampling period** command. The value is given in milliseconds.

Output format
This is the output format used for streamed or fetched data. See also the **outputformat** command.
- 00 **caltext01** format,
- 01 **caltext02** format,
- 02 **caltext03** format,
- 03 **caltext04** format

Logger status
This is the logger's status when it was successfully enabled, and so can have only a small number of values: 1 = **pending**, 2 = **logging**, or 4 = **gated**. See also the **status** command.

Serial link baudrate
This is the baudrate in force on the logger's serial link when the logger was enabled.

Feature flags
This is a bitmask of user selectable features which were active at the time the logger was enabled.
0x00000001 1 = logger prompt turned on.
0x00000002 1 = logger confirmations turned on.
0x00000004 1 = stream sample data to USB port.
0x00000008 1 = stream sample data to serial port.
0x00000010 1 = store average of burst.
0x00000020 1 = store all measurements in burst.
0x00000040 1 = store tidal average of burst.
0x00000080 1 = store wave burst.
0x00000100 1 = fast (>1Hz) continuous sampling.
0x00004000 1 = sampling was gated by thresholding feature.
0x00008000 1 = logger keeps all sensor channels powered up between samples.
0x00010000 1 = sampling was gated by the twist activation feature.
0x00020000 1 = the regimes sampling mode was enabled.
0x00040000 1 = cast detection for profiling deployments was enabled.
0x00080000 1 = the serial port auxiliary control feature was enabled.
0x00100000 1 = logger was enabled in simulated data mode (added for header version 1.011).
0x00200000 1 = the directional dependent sampling mode was enabled (added for header version 1.014).

Average interval
Measured in milliseconds, this is the time between averaged bursts.

Average length
A count of the number of measurements in an averaged burst.

Burst interval
Measured in milliseconds, this is the time between recorded bursts.

Burst length
A count of the number of measurements in a recorded burst.

Altitude
This item is a floating point number in IEEE 32-bit (single precision) format, not an integer. It represents the height above the sea bed at which the logger is deployed. The units are not specified, although RBR Ltd's host software Ruskin expects the value to be in metres. This parameter is meaningful only for instruments configured to record wave bursts; see also the **altitude** command.

Thresholding channel
The channel monitored for threshold-gated sampling, if enabled (see Feature flags above). The first channel has an index of 1.

Thresholding condition
The condition which must be satisfied for the logger to record data if threshold-gated sampling was enabled (see the Feature flags above). A value of 0 is used when the reading must be below the threshold for sampling to occur, a value of 1 is used if the reading must be above the threshold.

Thresholding value
This item is a floating point number in IEEE 32-bit (single precision) format, not an integer. It is the threshold value in calibrated units, used for comparison with values from the monitored channel, if threshold-gated sampling was enabled (see the Feature flags above).

Thresholding interval
This is the interval in milliseconds between threshold checks when the logger is in the **gated** state, waiting for a threshold trigger.

Fetch power off delay
This is the timeout in milliseconds before the sensors are powered off after sending the fetch command.

Temperature default value
This item is a floating point number in IEEE 32-bit (single precision) format, not an integer.

Conductivity default value
This item is not in use anymore; from Header Version 1.012 and onward it is replaced by the **Salinity default value**.

Pressure default value
This item is a floating point number in IEEE 32-bit (single precision) format, not an integer.

Atmospheric pressure default value
This item is a floating point number in IEEE 32-bit (single precision) format, not an integer.

Density default value
This item is a floating point number in IEEE 32-bit (single precision) format, not an integer.

Regimes settings
bit 7 : 0 = ascending, 1 = descending
bit 6 : 0 = absolute pressure reference, 1 = sea pressure reference
bits 0..5 : number of regimes (between 1 and 3)

Regime 1 boundary (2 bytes)
The boundary is in dbar.

Regime 1 binsize (2 bytes)
The binsize is in dbar.

Regime 1 period
The sampling period during this regime in milliseconds

Regime 2 boundary (2 bytes)
The boundary is in dbar.

Regime 2 binsize (2 bytes)
The binsize is in dbar.

Regime 2 period
The sampling period during this regime in milliseconds

Regime 3 boundary (2 bytes)
The boundary is in dbar.

Regime 3 binsize (2 bytes)
The binsize is in dbar.

Regime 3 period
The sampling period during this regime in milliseconds.

Firmware type
The firmware type code as an integer; see the **fwtype** parameter under the id command.

Serial mode
The operating mode of the Serial port as an integer code; see the **mode** parameter under the serial command:

0: (**rs232**) standard RS-232.
1: (**rs485f**) full duplex RS-485.
2: (**uart**) direct connection to UART 3V logic.
3: (**uart_idlelow**) inverted connection to UART 3V logic.
4: (**rs485h**) half duplex RS-485.

Auxiliary serial control output; polarities
Determines the active polarity of the signal when the logger is awake, and the state of the signal when the logger is asleep:

b0: 0 = logger drives signal low to activate, 1 = logger drives signal high to activate.
b1: 0 = signal is low when logger is asleep, 1 = signal is high when logger is asleep.
b2: 0 = b1 controls signal level when logger is asleep, 1 = signal is high impedance when logger is asleep.

Auxiliary serial control output; setup time
The time in milliseconds for which the control signal is activated before the logger starts to stream data over the Serial link.

Auxiliary serial control output; hold time
The time in milliseconds for which the control signal is held active after the logger has finished streaming data over the Serial link.

Reference pressure for WiFi module, if installed

This item is a floating point number in IEEE 32-bit (single precision) format, not an integer. It represents the pressure in dbar assumed to apply at the surface of the water at the start of the deployment.

Power-on timeout for WiFi module, if installed

After activating the WiFi module, the logger will wait for this number of seconds at the most for a valid command to be received. If the timeout expires, the WiFi module will be deactivated again.

Command timeout for WiFi module, if installed

After receiving a valid command, the logger will wait for this number of seconds at the most for another. If the timeout expires, the WiFi module will be deactivated.

Offset from Universal Coordinated Time (UTC)

This parameter was added in Header Version 1.009. It represents an offset in hours from UTC in the local timezone when the logger was deployed, if that information was provided by the host software. This item is a floating point number in IEEE 32-bit (single precision) format, not an integer. This format allows offsets which include partial hours, for those time zones which need it. If the timezone is never explicitly set, or is erased by using the clock (page 31) command to update the date/time, it will be reported as a NaN ('Not a Number' in IEEE floating point format).

Specific Conductivity Temperature Coefficient

This parameter was added in Header Version 1.009. It is used when correcting conductivity readings to 25 °C for the specific conductance derived channel. The value of the coefficient varies slightly depending on the ionic composition of the water, and is typically in the range 0.0191 to 0.0214. The default value is 0.0191, suitable for standard KCl solutions . This item is a floating point number in IEEE 32-bit (single precision) format, not an integer .

Simulated data cycle period

This parameter was added in Header Version 1.011. When a logger is enabled in simulation mode, the simulated data is periodic, with the period udually being much longer than the sampling period. The cycle period of the simulated data is given as an unsigned 32-bit integer in milliseconds. For example, a value of 3600000 represents a period of one hour .

Salinity default value

Added in Header Version 1.012, replacing the **Conductivity default value**; t his item is a floating point number in IEEE 32-bit (single precision) format, not an integer.

Average sound speed default value

Added in Header Version 1.013; this item is a floating point number in IEEE 32-bit (single precision) format, not an integer.

Directional dependent sampling flags

This is a bitmask of settings for directional dependent sampling (a dded in Header Version 1.014)
0x00000001 1 = ascending direction.

Directional dependent fast sampling period

The fast sampling period in milliseconds (a dded in Header Version 1.014)

Directional dependent slow sampling period

The slow sampling period in milliseconds (a dded in Header Version 1.014)

Directional dependent fast sampling threshold

The fast sampling threshold in dbar, this item is a floating point number in IEEE 32-bit (single precision) format, not an integer (a dded in Header Version 1.014)

Directional dependent slow sampling threshold

The slow sampling threshold in dbar, this item is a floating point number in IEEE 32-bit (single precision) format, not an integer (a dded in Header Version 1.014)

Cha nnel section

Type : 0x03 (1 byte)

Length : (2 bytes) The size of this section is not fixed, it depends on the number and type of channels. Every logger includes at least one channel.

Data structure : *This structure size is not a fixed size. It depends on how many channels are in the logger. , *Following the parameters is 181 bytes of padding (0xFF values)

Number of channels
As reported by the **channels** command, (1 byte)

Channel 1 offset
Channel 1 details offset from beginning of the channel section structure, (2 bytes)

Channel 2 offset (2 bytes)

...

Channel N offset (2 bytes)

[Channel_1 details structure] (variable size, depends on channel type and number of calibration coefficients)

[Channel_2 details structure] (variable size...)

...

[Channel_N details structure] (variable size...)

Channel details structure

Type (6 bytes)
This is a 6-character ASCII string as reported by the **channel** command. See ***Supported channel types*** for a list of possible values

Channel extensions (2 bytes)
This 16-bit field stores specific flags for the channel.
The first five bits are used internally by the logger to control the properties and behaviour of a channel which has been turned off using the **channel status** command. The bits are assigned as follows:

- - bit-0: 0 if the channel is visible, 1 if it is hidden.
  - bit-1: 0 if the channel is sampled, 1 if it is ignored.
  - bit-2: 0 if the channel data is stored in memory, 1 if it is transient.
  - bit-3: 0 if the channel data may be streamed, 1 if it is quiet.
  - bit-4: 0 if the channel is "on", 1 if the user requested it be turned "off".

These bits are not directly accessible to users via the command interface, and are really only for the logger's internal use: not all possible combinations are valid. Turning a channel "off" is not as simple as it may seem; for example, if the user requests a channel is turned off, but its data is required for the correction of another channel, the logger must still sample the channel without necessarily storing or streaming the data. These bits provide the detailed control necessary to deal with such scenarios.

The visible/hidden property is configured at the Factory and can not be changed or read by the user. A channel may be hidden if it is required by the logger but is of no interest to the user; an example might be an internal temperature needed for correction purposes.

Calibration Date (4 bytes)
Retrieved by the **calibration** command. The value is the number of seconds elapsed since 2000-Jan-01 00:00:00.

Number of coefficients (1 byte)
The number of channel calibration coefficients.

Coefficient 1 (4 bytes)
This item is typically a floating point number in IEEE 32-bit (single precision) format, not an integer.

Coefficient 2 (4 bytes)

...

Coefficient N (4 bytes)

All true coefficients are floating point numbers in IEEE 32-bit (single precision) format, but he channel indices for cross-channel correction or compensation (the **n**-group) are 32-bit integers. Refer to the **calibration** command for more details.

Ranging mode (1 byte)
This is the status of the gain control on the sensor card. It can have only one of three values: 0 = **none** , 1 = **manual** , 2 = **auto** . See also the **channel <index> gain** command. If the mode is set to **none** then the values for the rest of the gain parameters are just padded with 0x00 and shouldn't be used for anything.

Number of available gains (1 byte)
This is the number of possible gains that can be used for manual or auto ranging modes. If the ranging mode is 0 then this value will be 0 (padding) otherwise the possible values are 1 – 4. If this value is less than the maximum (4) then the remaining **Available gains** are padded with 0x00.

Current Gain value (4 bytes)
This is the gain currently in use at time of deployment. If the channel is in ranging mode 0 then this value is 0 and should be ignored. If the ranging mode is 2 then this value is the first gain in the **Available gain 1** . If the ranging mode is 1 then this value will be the manual gain setting chosen by the user at the time of deployment. This item is a floating point number in IEEE 32-bit (single precision) format, not an integer.

Available gain 1 (4 bytes)
This is one possible gain value available to the sensor. If ranging mode is **none** then this value is padded with 0x00. This item is a floating point number in IEEE 32-bit (single precision) format, not an integer.

Available gain 2 (4 bytes)
This is one possible gain value available to the sensor. If ranging mode is **none** or the **Number of available gains** < 2 then this value is padded with 0x00. This item is a floating point number in IEEE 32-bit (single precision) format, not an integer.

Available gain 3 (4 bytes)
This is one possible gain value available to the sensor. If ranging mode is **none** then or the **Number of available gains** < 3 then this value is padded with 0x00. This item is a floating point number in IEEE 32-bit (single precision) format, not an integer.

Available gain 4 (4 bytes)
This is one possible gain value available to the sensor. If ranging mode is **none** then or the **Number of available gains** < 4 then this value is padded with 0x00. This item is a floating point number in IEEE 32-bit (single precision) format, not an integer.

Front-end specific structures total size (2 bytes)
Except for BPR frontends, and for channels which support the sensor (page 91) command, this is set to zero. This is followed by a serie of front-end specific structures. The **Front-end specific structures total size** is the total size of all the following **Front-end specific structure** .

Front-end specific structure
Applies to BPR frontends.

Also applies to channels which support the sensor (page 91) command.

In the case of a channel supporting the sensor (page 91) command, for each parameter returned by the sensor command (key/value), there will be one sensor key/value structure.This structure starts with a short 4-byte header, which is followed by 1 pair of <*name*>,<*value*> entry, as follows:

type (1 byte), an integer indicating the type of information stored in <*value*>; presently this is always 2, indicating a string key/value pair.
size (1 byte), gives the size of the data area containing the <*name*>,<*value*> entry, not including the 4-byte header itself. The size is measured in 4-byte (32-bit) units (*not* 8-bit bytes).
offset (2 bytes), Presently unused, and always set to 0.

Then come the pairs of <*name*>,<*value*> entries.

name, stored as a sequence of lower-case ASCII characters, ending with a NUL character (byte value of 0). This may be padded after the NUL if necessary with up to three bytes of value 127 (0x7F), so that the total size of the name entry in bytes is a multiple of four.
value, the size and format of which will depend on the **type** entry in the header; presently only strings are supported, stored as a sequence of lower-case ASCII characters, ending with a NUL character (byte value of 0).This may be padded after the NUL if necessary with up to three bytes of value 127 (0x7F), so that the total size of the name entry in bytes is a multiple of four.

There is no padding between the header and the first entry, or between consecutive entries; in total the entries will occupy the number of 4-byte units given in the **size** entry of the header.

5.3.2 Sample data standard format

Following the deployment header in memory is the deployment data, which will contain both sample data and events, recorded in chronological order. Events are time-stamped records of any notable incidents apart from sample data. Both sample data and events have their own formats; refer to the following sections for details.

Sample data *should* always be stored as complete sets of readings, one set comprising one reading from each active channel, all taken at the same time. However, it may be possible under some fault conditions for only a partial sample set to be stored. It is therefore important when parsing data to check for, and be able to identify, event markers at every reading, not just at the assumed start of every sample set. However, partially stored sample sets are extremely rare, and for the remainder of this discussion we will assume for simplicity that all sample sets are complete.

Sample timing

Sample sets are not individually time-stamped; only events contain any explicit date and time information. This avoids 'wasting' memory by time-stamping samples which are regular and predictable, but the date and time must be calculated by counting sample sets from the previous time-stamped event, and applying the programmed parameters for the schedule, such as the measurement period and the burst interval, if applicable: refer to the **sampling** command for details. At the very least, there should always be one time-stamped event in the deployment; immediately after the header and before the first sample set.

Normal reading values

Each individual reading of the sample data is a signed 32-bit integer in 2's-complement format, in principle giving a range from –2147483648 to +2147483647. However, not all of this range is fully utilized for sample data; in particular, some parts of the range have been assigned for purposes such as event markers and error codes.

Individual reading values at present are typically confined to a sub-range of the full 32-bit range available, namely 134217728 to +1073741760. Readings outside this range which are not:

1. defined as Event Markers (see the section Standard format events markers (page 133)), or
2. defined as Error Codes (see below), or
3. known to originate from a sensor channel with specially defined properties,

should be treated with suspicion.

The readings for each channel are 'raw', unprocessed values from an A/D converter or some other type of data acquisition hardware. They can not be interpreted as physical values of the measured parameter without further processing according to the calibration equation and coefficients for the channel.

Error Codes

Under some conditions an error may occur on one channel while data from the other channels is perfectly acceptable. Rather than inserting a time-stamped event in the data stream if this happens, the individual reading is replaced by an error code.

Error codes are 32-bit values which by definition should never appear as valid readings. They indicate a problem with that particular reading from the channel in question; other readings in the same sample set may be fine, as may other readings from the same channel in different sample sets.

The general format is 0xF6<*EC*><*CRC*>, where 0xF6 is the error code indicator in the MSB, <*EC*> is one of 256 possible error codes, and <*CRC*> is the byte-swapped 16b CRC of the two bytes 0xF6,<*EC*>. Because the codes are all fixed, so are the CRCs, and the table below simply shows the full 32-bit values for all error codes defined so far.

0	0xF600D692	generic, unknown or unexpected error
1	0xF601E7A1	EOC bit unexpectedly set in ADC output
2	0xF602B4F4	DMY bit unexpectedly set in ADC output
3	0xF60385C7	internal addressing error
4	0xF604125E	too much data for internal transfer
5	0xF605236D	access to internal bus denied
6	0xF6067038	timeout sending internal command
7	0xF607410B	timeout receiving internal response
8	0xF6087F1B	generic failure to interpret response
9	0xF6094E28	no sample was started
10	0xF60A1D7D	sample acquisition still in progress

11	0xF60B2C4E	sample process failed
12	0xF60CBBD7	no valid samples to average
13	0xF60D8AE4	internal response unexpectedly short
14	0xF60ED9B1	supporting channel value not valid, or unknown equation
15	0xF60FE882	(reserved)
16	0xF610A591	channel value is outside reasonable range
17	0xF61194A2	channel value is below minimum measurable limit
18	0xF612C7F7	channel value is above maximum measurable limit

5.3.3 Standard format events markers

Events are records of non-sample incidents, and can be used to aid interpretation of the deployment data, or for diagnostic purposes.

In Standard format, events and sample data are stored together in chronological order in dataset-1. Sample data should always be stored as complete sets of readings, one set comprising one reading from each active channel, all taken at the same time. However, it may be possible under some fault conditions for only a partial sample set to be stored. It is therefore important when parsing data in Standard format to check for event markers at every *reading*, not just at the assumed start of every sample set.

Event structure

Bytes	Description
Bytes 0,1	16-bit byte-swapped CCITT CRC of Bytes 2..7
Byte 2	Type Code
Byte 3	0xF3 marker byte
Bytes 4..7	32-bit date/time in elapsed seconds format

Bytes	Description
Bytes 8..9	16 b count of milliseconds in current second [0..999]
Byte 10	Size of event in uint32_t N
Byte 11	Event processing info
Bytes 12..4*N-1	Auxiliary data

Seconds for the date/time are counted from 2000-Jan-01 00:00:00.

Event processing info byte

Bit #	Description
bit 0	1 - the timestamp of this event is the timestamp of the next following sample set. 0 - the timestamp of this event does not affect the timestamp of the next following sample set.
bit 1..7	Unused

Type Codes

Type code	Description
0x00	Unknown or unrecognized events
0x01	Time synchronization marker
0x02	**stop** command received
0x03	Run-time error encountered
0x04	CPU reset detected

Type code	Description
0x05	One or more parameters recovered after reset
0x06	Restart failed : Real Time Clock/Calendar contents not valid
0x07	Restart failed : logger status not valid
0x08	Restart failed : primary schedule parameters could not be recovered.
0x09	Unable to load alarm time for next sample
0x0A	Sampling restarted after resetting Real Time Clock (RTC)
0x0B	Parameters recovered, sampling restarted after resetting RTC
0x0C	Sampling stopped, end time reached
0x0D	Start of a recorded burst
0x0E	Start of a wave burst
0x0F	(reserved)
0x10	Streaming now OFF for both ports
0x11	Streaming ON for USB, OFF for serial
0x12	Streaming OFF for USB, ON for serial
0x13	Streaming now ON for both ports
0x14	Sampling started, threshold condition satisfied
0x15	Sampling paused, threshold condition not met
0x16	Power source switched to internal battery
0x17	Power source switched to external battery

Type code	Description
0x18	Twist activation started sampling
0x19	Twist activation paused sampling
0x1A	WiFi module detected and activated
0x1B	WiFi module de-activated; removed or activity timeout
0x1C	Regimes enabled, but not yet in a regime
0x1D	Entered regime 1
0x1E	Entered regime 2
0x1F	Entered regime 3
0x20	Start of regime bin
0x21	Begin profiling 'up' cast
0x22	Begin profiling 'down' cast
0x23	End of profiling cast
0x24	Battery failed, schedule finished.
0x25	Directional dependent sampling, beginning of fast sampling mode
0x26	Directional dependent sampling, beginning of slow sampling mode

Auxiliary data

Not all the events have embedded auxiliary data; here is a comprehensive list of those that do, with descriptions of the embedded data.

Regime bin event 0x20 auxiliary data

Bytes	Description
12..15	Number of readings in the bin

Firmware versions 1.100 or later may also support a derived data channel, type **cnt_00** , which contains this value from event 0x20 when in the regimes sampling mode. Refer to the Section "Integrating with a profiling float (page 18) " for further details.

Begin profiling 'up' cast event auxiliary data

Bytes	Description
12..15	32-bit (4-byte) address of the corresponding sample set.

Begin profiling 'down' cast event auxiliary data

Bytes	Description
12..15	32-bit (4-byte) address of the corresponding sample set.

End of profiling cast event auxiliary data

Bytes	Description
12..15	32-bit (4-byte) address of the first sample set after the cast.

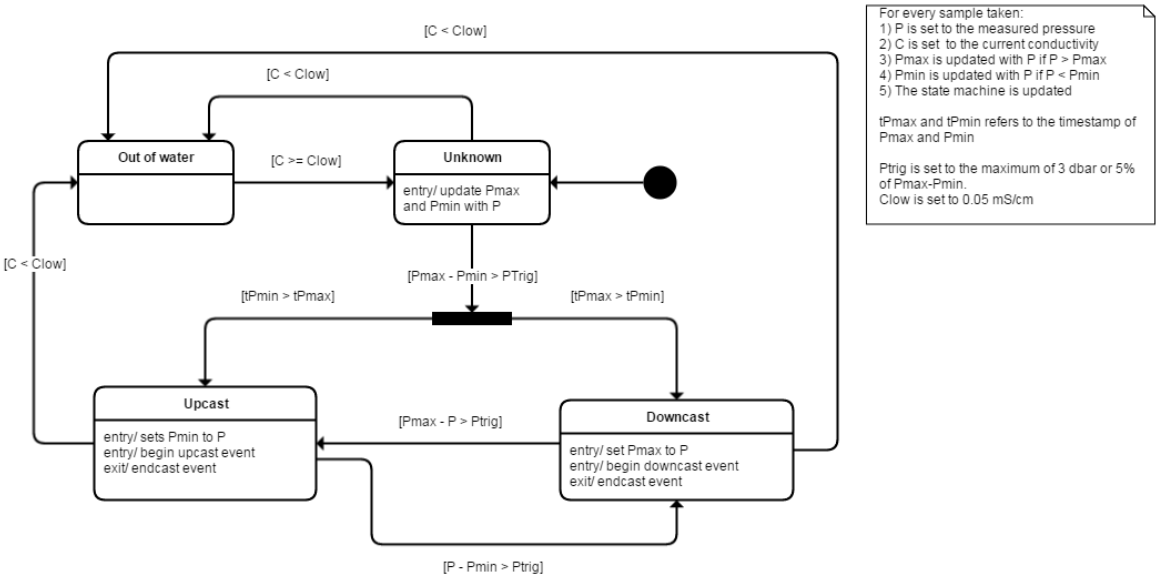
Runtime error event auxiliary data

Bytes	Description
12..15	32-bit (4-byte) firmware program address at which the error was detected

5.4 Profile detection events generation

When the logger is configured with settings castdetection = on (page 83), it generates cast events in the recorded data (see EasyParse format events markers (page 116) and Standard format events markers (page 133)).
There are basically three types of cast events: beginning of an upcast, beginning of a downcast and end of cast.

The following state machine is used to determine those events.



6 Supported Channel Types

The following is a list of the channel types supported at the time of writing this document. These type names are used by the channel (page 78) command.

Type	Equation	Units	Manufacturer	Description
acc_00	linear	g	RBR	Acceleration (in g's)
alti00	distancefromechotiming	m	RBR	Distance from echo timing
bpr_00	linear	ps	Paroscientific /RBR	BPR pressure period
bpr_01	linear	ps	Paroscientific /RBR	BPR temperature period
bpr_02	linear	dbar	Paroscientific /RBR	BPR calculated pressure
bpr_03	linear	°C	Paroscientific /RBR	BPR calculated temperature
bpr_04	linear	ps	Paroscientific /RBR	BPR pressure period
bpr_05	linear	ps	Paroscientific /RBR	BPR temperature period
bpr_06	linear	dbar	Paroscientific /RBR	BPR calculated pressure
bpr_07	linear	°C	Paroscientific /RBR	BPR calculated temperature
bpr_08	deri_bprpres	dbar	Paroscientific /RBR	BPR calculated pressure

Type	Equation	Units	Manufacturer	Description
bpr_09	deri_bprtemp	°C	Paroscientific /RBR	BPR calculated temperature
cnt_00	none	counts	RBR	Measurement count, useful for eg. regimes bins
cond03	corr_cond	S/cm	RBR	Fresh water conductivity (corrected, 12Hz)
cond06	corr_cond	mS/cm	RBR	Marine conductivity (corrected, 12Hz)
cond07	corr_cond	mS/cm	RBR	Marine conductivity (corrected, 12Hz, Combined C.T cell)
doxy03, doxy11	linear	%	Oxyguard	Dissolved oxygen saturation (Note 2)
doxy01	linear	%	Aanderra	Optode: dissolved oxygen saturation, analogue
doxy02	corr_rinko	%	JFE Advantech	Rinko-III: dissolved oxygen saturation with temperature
doxy06	corr_rinkoT	%	JFE Advantech / RBR	Rinko-III: dissolved oxygen saturation without temperature
doxy07	linear	%	Aanderaa	Optode: dissolved oxygen saturation, serial
doxy08	corr_rinkoB	%	JFE Advantech	Rinko-III (B): dissolved oxygen saturation with temperature

Type	Equation	Units	Manufacturer	Description
doxy09	corr_rinkoBT	%	JFE Advantech / RBR	Rinko-III (B): dissolved oxygen saturation without temperature
doxy10	linear	mol/L	Aanderaa	Optode: dissolved oxygen concenration, serial
doxy13	linear	%	RBR	Dissolved oxygen saturation, serial RBR*solo*. DO|rt
doxy22	deri_o2sat_garcia	%	RBR	Dissolved oxygen saturation derived from concentration via Gordon and Garcia
doxy23	corr_o2conc_garcia	mol/L	RBR	Dissolved oxygen concentration compensated, serial RBR *coda* ODO
doxy27	corr_o2conc_garcia	mol/L	RBR	Dissolved oxygen concentration compensated, serial RBR *coda* ODO|slow
dpth01	deri_depth	m	-	Derived depth
echo01	lin	ms	RBR	Echo timing
fluo00	linear	g/L	Seapoint	Fluorometry-Phycoerythrin
fluo01	linear	g/L	Seapoint	Fluorometry-Chlorophyll
fluo02	linear	g/L	Seapoint	Fluorometry-Rhodamine
fluo03	linear	g/L	Seapoint	Fluorometry-UV/CDOM

Type	Equation	Units	Manufacturer	Description
fluo04	linear	g/L	Seapoint	Fluorometry-Phycocyanin
fluo10	linear	g/L	Turner Designs	Fluorometry-Chlorophyll-a
fluo11	linear	ppb	Turner Designs	Fluorometry-CDOM
fluo12	linear	ppb	Turner Designs	Fluorometry-Crude oil
fluo13	linear	cells/mL	Turner Designs	Fluorometry-Cyanobacteria
fluo14	linear	ppb	Turner Designs	Fluorometry-Optical brighteners
fluo15	linear	ppb	Turner Designs	Fluorometry-Fluorescein
fluo16	linear	ppb	Turner Designs	Fluorometry-Rhodamine
fluo17	linear	ppb	Turner Designs	Fluorometry-Refined fuels
fluo18	linear	ppm	Turner Designs	Fluorometry-BTEX
fluo19	linear	cells/mL	Turner Designs	Fluorometry-Phycocyanin
fluo20	linear	cells/mL	Turner Designs	Fluorometry-Phycoerythrin
fluo21	linear	V	Turner Designs	Fluorometry-custom

Type	Equation	Units	Manufacturer	Description
fluo22	linear	RFUB	Turner Designs	Chlorophyll a (C3)
fluo23	linear	RFUB	Turner Designs	CDOM (C3)
fluo24	linear	RFUB	Turner Designs	Crude oil (C3)
fluo25	linear	RFUB	Turner Designs	Cyanobacteria (C3)
fluo26	linear	RFUB	Turner Designs	Optical brighteners (C3)
fluo27	linear	RFUB	Turner Designs	Fluorescein dye (C3)
fluo28	linear	RFUB	Turner Designs	Rhodamine dye (C3)
fluo29	linear	RFUB	Turner Designs	Refined fuels (C3)
fluo30	linear	RFUB	Turner Designs	BTEX (C3)
fluo31	linear	RFUB	Turner Designs	Phycocyanin (C3)
fluo32	linear	RFUB	Turner Designs	Phycoerythrin (C3)
fluo33	linear	counts	WET Labs	Chlorophyll a (ECO Puck)
fluo34	linear	counts	WET Labs	CDOM (ECO Puck)

Type	Equation	Units	Manufacturer	Description
fluo35	linear	counts	WET Labs	Phycoerythrin (ECO Puck)
mag_00	linear	T	RBR	Magnetic field strength (in micro-Tesla's)
meth00	corr_metsmet	mol/L	Franatech	METS methane concentration
opt_07	linear	°	RBR	Calibrated phase output from RBR*coda* ODO
opt_14	linear	°	RBR	Calibrated phase output from RBR*coda* ODO|slow
orp_00	linear	V	AMT	ORP (Oxidation/Reduction Potential)
orp_01	linear	V	Idronaut	ORP (Oxidation/Reduction Potential)
par_00	linear	mol/m²/s	Licor	PAR (photosynthetically active radiation)
par_01	linear	mol/m²/s	Biospherical	PAR (photosynthetically active radiation)
pco200	linear	ppm	Turner Designs	C-sense CO2 partial pressure
peri00	linear double	ps	RBR	Frequency counter
peri01	linear double	ps	RBR	Frequency counter
ph__00	linear	pH_units	AMT	pH
ph__01	linear	pH_units	Idronaut	pH

Type	Equation	Units	Manufacturer	Description
ph__02	corr_ph	pH_units	Idronaut	Corrected pH
phas00	linear	degrees	Aanderaa	Calibrated phase output from serial AADI Optode
pres19	corr_pres2	dbar	Keller / generic	Pressure (absolute, temperature compensated)
pres23	linear	dbar	RBR	Pressure (absolute, temperature compensated), serial RBR *solo*.D|rt, RBR*duet*.T.D|rt
pres08	deri_seapres	dbar	-	Derived hydrostatic (sea) pressure
sal_00	deri_salinity	PSU	-	Derived salinity, PSS78.
scon00	deri_speccond	S/cm	-	Derived specific conductivity
sos_00	deri_sos	m/s	-	Derived speed of sound, UNESCO, Chen and Millero
temp04, temp09	temp	°C	RBR	Temperature (Note 2)
temp01	corr_rinkotemp	°C	JFE Advantech	Temperature, Rinko-III
temp06	corr_metstemp	°C	Franatech	Temperature, METS
temp07	linear	°C	Aanderaa	Temperature, from serial Optode
temp11	temp	°C	RBR	

Type	Equation	Units	Manufacturer	Description
				Temperature, compensating channel for conductivity
temp13	linear	°C	RBR	Temperature, serial RBR *solo*.D|rt, RBR*duet*.T.D|rt
temp14	temp	°C	RBR	Temperature, Combined C. T cell
temp16	linear	°C	RBR	Temperature, serial RBR *coda* ODO
temp17	linear	°C	RBR	Temperature, serial RBR *coda* ODO|slow
tran00	linear	trans_units	generic	Transmittance (Note 1)
tran01	linear	%	WET Labs	Transmittance
tran02	linear	V	WET Labs	Transmittance
turb00	linear	NTU	Seapoint	Turbidity
turb01	linear	NTU	Turner Designs	Turbidity
turb02	linear	NTU	Campbell	Turbidity, OBS3+
turb03	linear	NTU	Turner Designs	Turbidity, C3 serial
turb04	linear	counts	WET Labs	Backscatter, ECO Puck
volt00	linear	V	generic	Voltage (0V to +5V)
volt01	linear	V	generic	Voltage (–2.5V to +2.5V)

Type	Equation	Units	Manufacturer	Description
volt02	linear	V	generic	Voltage (0V to +10V)
volt03	linear	V	generic	Voltage (–10V to +10V)

Notes

1. Includes SeaTech and WET Labs.
2. The following channel types within the listed groups differ only in their internal implementation; for end users there is no distinction between them:
   - **doxy03**, **doxy11**
   - **temp04**, **temp09**
3. **bpr_04**,**bpr_05**,**bpr_06**,**bpr_07, bpr_08, bpr_09, peri00, peri01** are calculated using double precision in order to maintain resolution.

7 Calibration Equations and Cross-channel Dependencies

The primary input to most equations is R, a raw number normalized to a nominal full scale of 1. This is typically a binary reading from an A/D converter divided by a full scale value of 2^30, and so is often referred to as a 'voltage ratio', but other input hardware and scale factors may be used.

A few equations for derived parameters use only secondary inputs from other channels; they have no underlying measurement hardware, and so no R input. A good example would be salinity, which is a function of conductivity, temperature, and pressure.

7.1 Core Equations

The logger presently supports four 'core' equations with no cross channel dependencies, so they use only the **c** group of coefficients; there are no terms using the **x** or **n** groups.

7.1.1 lin, or Linear

```
output = c0 + c1*R
```

7.1.2 qad, or Quadratic

```
output = c0 + c1*R + c2*R^2
```

7.1.3 cub, or Cubic

```
output = c0 + c1*R + c2*R^2 + c3*R^3
```

7.1.4 tmp, or Temperature

Given in °C, based on the Steinhart-Hart equation used for thermistors.

```
T = (1/Y) - 273.15
```

where

```
Y = c0 + c1*X + c2*X^2 + c3*X^3
```

and

```
X = ln(1/R - 1)
```

7.2 Specialized Equations

The logger also supports another group of equations with no cross channel dependencies, but they are typically used for only a single type of sensor and have no general purpose application. Depending on the complexity of the equation, they may use both **c**-group and **x**-group coefficients, but there are no cross channel reference indices in the **n**-group.

7.2.1 corr_rinkotemp - Temperature measured by a Rinko DO sensor

RBR data loggers support the integration of the Rinko-III dissolved oxygen (DO) sensor from JFE Advantech. In addition to the raw DO sensing element, this sensor also provides an output representing temperature, for the purpose of compensating the temperature dependence of the DO output. That compensation is described in a later section (see Example 5: corr_rinko - Correction of Rinko Dissolved Oxygen using Rinko Temperature sensor (page 159)); this section describes how temperature is derived from the Rinko-III temperature output, which requires its own channel in the logger in order to be monitored.
This is not strictly a 'corrected' output channel, as it has no dependence on any other channel in the logger. However it does make use of the **x**-group of coefficients, commonly used by corrected channels, to describe the behaviour of the logger electronics, whereas the primary coefficients in the **c**-group represent the behaviour of the sensor itself; it is useful to keep them in separate groups.
The equation is:

```
T = c0 + c1*V + c2*V^2 + c3*V^3
```

where

```
V = x0 + x1*R
```

The output of the temperature sensor is a voltage V, which is related to the logger's reported voltage ratio R by a simple linear equation, the coefficients of which (**x0**,**x1**) are determined by RBR Ltd at the factory. The primary coefficients **c0**...**c3** are provided by JFE Advantech for the cubic polynomial which relates the voltage V to the temperature output T in °C.

Example commands:

```
>> calibration 5 type
<< calibration 5 type = temp01
```

(Confirm the channel type)

```
>> calibration 5 datetime = 20130401120000, c0 = -5.65608, c1 = 16.80047, c2 =
-2.253705, c3 = 0.4827284
```

(Set the core coefficients for the temperature output)

```
>> calibration 5 datetime = 20130401120005, x0 = 6.782656, x1 = -9.257345
```

(Set the secondary coefficients for the voltage conversion)

```
>> calibration 5
<< calibration 5 type = temp01, datetime = 20130401120005, c0 = -5.65608, c1 =
16.80047, c2 = -2.253705, c3 = 0.4827284, x0 = 6.782656, x1 = -9.257345
```

(Request confirmation of everything)

7.2.2 corr_metstemp - Temperature measured by a METS (methane sensor)

RBR data loggers support the integration of the METS methane sensor from Franatech. In addition to the raw methane sensing element, this sensor also provides an output representing temperature, for the purpose of compensating the temperature dependence of the methane output. That compensation is described in a later section (see Example 10: corr_metsmeth - Temperature correction of METS methane output (page 167)); this section describes how temperature is derived from the METS temperature output, which requires its own channel in the logger in order to be monitored.
This is not strictly a 'corrected' output channel, as it has no dependence on any other channel

in the logger. However it does make use of the **x**-group of coefficients, commonly used by corrected channels, to describe the behaviour of the logger electronics, whereas the primary coefficients in the **c**-group represent the behaviour of the sensor itself; it is useful to keep them in separate groups.
The equation is:

```
T = c0 + c1*V
```

where

```
V = x0 + x1*R
```

The output of the temperature sensor is a voltage V, which is related to the logger's reported voltage ratio R by a simple linear equation, the coefficients of which (**x0**,**x1**) are determined by RBR Ltd at the factory. The primary coefficients **c0**,**c1** are provided by Franatech for the linear relation between the voltage V and the temperature output T in °C.

Example commands:

```
>> calibration 5 type
<< calibration 5 type = temp06
```

(Confirm the channel type)

```
>> calibration 5 datetime = 20130401120000, c0 = -5.65608, c1 = 16.80047
```

(Set the core coefficients for the temperature output)

```
>> calibration 5 datetime = 20130401120005, x0 = 6.782656, x1 = -9.257345
```

(Set the secondary coefficients for the voltage conversion)

```
>> calibration 5
<< calibration 5 type = temp06, datetime = 20130401120005, c0 = -5.65608, c1 = 16.80047, x0 = 6.782656, x1 = -9.257345
```

(Request confirmation of everything)

7.3 Dependent Equations

All other equations currently implemented in the logger involve cross-channel dependencies, and so are intrinsically more complicated. A cross-channel dependency exists if the output of a channel depends on raw data from more than one channel in the logger. Typically, there is a primary raw input from the channel in question, with one or more secondary inputs from other channels, but there are other variations.

The equation type used for such a channel has knowledge of these dependencies built into it, but needs to be told which channel(s) in this particular logger are to be used. For a given channel type, just as the values of the usual coefficients in the **c** group vary from one logger to another, so will the values of the coefficients in the cross-compensation group **x**.

In many cases it is both useful and feasible for a channel to be recalibrated by end users, and typically it is the **c** group coefficients which will need to be changed. The cross-compensation coefficients in the **x** group often do not vary significantly over time, and may be much harder to determine. Although it is possible for OEM users to modify the values, it is not recommended as routine practice; one reason for using a different name for these coefficients is to act as a warning cue against accidental modification.

The indices of the secondary input channels in the **n** group will also differ between loggers. For example, a temperature channel required for compensation may be Channel-1 in one logger, but Channel-3 in another. The index values are configured at the factory and can not be changed by users, but their values can be read.

There is one special case when the value of an **n** index may be the text field "value". Again, this can be set only at the factory, and applies when an equation requires a correction term using a parameter which the logger does not measure. In this case, the default parameter set by the command **settings** will be used.

The dependent equations are explained by example on the following pages. For a first reading, it is suggested they are studied in order, as new concepts are introduced progressively, and some of the later examples are more complicated.

7.3.1 Example 1: corr_pH - Simple temperature correction of pH

Consider an RBR*concerto* C.T.D.pH logger. Without temperature correction, the pH output from Channel-4 would be a simple linear function of the raw data:

```
pHraw = c0 + c1*R
```

where R is the normalized voltage ratio from Channel-4 monitoring the pH sensor,**c0,c1** are the core coefficients of the linear equation, andpHraw is the uncorrected output in pH units.

The parameter pH is well known to have a dependence on temperature, so a more accurate value is obtained if the compensated version of the equation is used. This is typically expressed in a form such as:

```
pHcorr = pHraw + Kph * (pHraw – pHcal) * (T – Tcal)
```

Casting this into the form used by the logger, corr_ph, is simple:

```
pHcorr = pHraw + x0 * (pHraw – x1) * (value(n0) – x2)
```

where

- pHraw is **c0** + **c1** * R as before, now an intermediate variable in the equation,

- **x0** corresponds directly in value to the constant 'Kph',
- **x1** is the calibration pH 'pHcal', generally 7.0,
- **x2** is the calibration temperature 'Tcal' in °C,
- **n0** is the index of the temperature channel (2 in this example), value(**n0**) is the final output value of the temperature channel in °C,
- pHcorr is the corrected output in pH units.

Note that this equation is for the output from Channel-4, so the source of the primary raw data R is implicitly Channel-4: there is no '**n**' index to specify where the raw data originates.

Example commands:

```
>> calibration 4 type
<< calibration 4 type = ph__02
```

(Confirm the channel type)

```
>> calibration 4 datetime = 20130401120000, c0 = 15.23461, c1 = –0.198743
```

(Set the core coefficients)

```
>> calibration 4 datetime = 20130401120005, x0 = –0.00302, x1 = 7, x2 = 24.943
```

(Set the cross-channel correction coefficients)

```
>> calibration 4
<< calibration 4 type = ph__02, datetime = 20130401120005, c0 = 15.23461, c1 =
–0.198743, x0 = –0.00302, x1 = 7, x2 = 24.943, n0 = 2
```

(Request confirmation of everything)

7.3.2 Example 2: corr_pH - pH correction without Temperature

In practice this is perhaps an unlikely scenario, and so a rather artificial example, but it is useful to use the simple equation for pH to illustrate the concept. Consider an RBR*concerto* monitoring conductivity and several electrochemical sensors, including pH. The deployment

conditions are already known to have a temperature which is approximately constant, so the instrument does not monitor temperature: perhaps the instrument is on the sea bed, and so at about 4°C all the time. But this is very different from the typical calibration temperature for pH of 25°C, so it would be desirable to correct the readings.

The compensated version of the equation itself is not changed:

```
pHcorr = pHraw + x0 * (pHraw – x1) * (value(n0) – x2)
```

but now there is no temperature channel, and so no value we can use for **n0**. In a case like this, **n0** will not have a numeric value of a logger channel index, but will be set at the factory to the special text string "**value**". The logger knows that a temperature value is required in the equation, and so will use the substitute temperature value "sub(T)" specified by the "**settings temperature**" command, instead of "value(**n0**)".

In this instance, the equation effectively takes the form below, and in the above example "sub (T)" might have a value such as 3.9 (in °C):

```
pHcorr = pHraw + x0 * (pHraw – x1) * (sub(T) – x2)
```

Example commands:

```
>> calibration 4 type
<< calibration 4 type = ph__02
```

(Confirm the channel type)

```
>> calibration 4 datetime = 20130401120000, c0 = 15.23461, c1 = –0.198743
```

(Set the core coefficients)

```
>> calibration 4 datetime = 20130401120005, x0 = –0.00302, x1 = 7, x2 = 24.943
```

(Set the cross-channel correction coefficients)

```
>> calibration 4
<< calibration 4 type = ph__02, datetime = 20130401120005, c0 = 15.23461,c1 =
–0.198743, x0 = –0.00302, x1 = 7, x2 = 24.943, n0 = value
```

(Request confirmation of everything)

```
>> settings temperature = 3.9
<< settings temperature = 3.9
```

(Set the value of the substitute temperature)

7.3.3 Example 3: corr_pres2 - Temperature correction of Pressure

Returning to the example of an RBR*concerto* C.T.D.pH logger, the pressure reading from Channel-3, without correction for the effect of temperature on the sensor, is given by a cubic polynomial:

```
Praw = c0 + c1*R + c2*R^2 + c3*R^3
```

where R is the normalized voltage ratio from Channel-3 monitoring pressure, **c0...c3** are the core coefficients of the cubic polynomial equation, and Praw is the uncorrected pressure output, reported in dbar for RBR instruments.

The equation which accounts for residual temperature sensitivity of the pressure sensor is:

$$P_{corr} = P_{cal} + \frac{(P_{raw} - P_{cal}) - K_{p1}(T - T_{cal}) - K_{p2}(T - T_{cal})^2 - K_{p3}(T - T_{cal})^3}{1 + K_{p4}(T - T_{cal})}$$

Casting this into the form used by the logger would yield:

$$P_{corr} = x_0 + \frac{(P_{raw} - x_0) - x_1(value(n_0) - x_5) - x_2(value(n_0) - x_5)^2 - x_3(value(n_0) - x_5)^3}{1 + x_4(value(n_0) - x_5)}$$

where

- Praw is the cubic polynomial in R, as before,
- **x0** is the calibration pressure 'Pcal' in dbar,
- **x1**, **x2, x3, x4** correspond directly to the constants 'Kp1' through 'Kp4',
- **x5** is the calibration temperature 'Tcal' in °C,
- **n0** is the index of the temperature channel, in this example 2,
- value(**n0**) is the final output value of the temperature channel in °C,
- Pcorr is the corrected output in dbar.

Example commands:

```
>> calibration 3 type
<< calibration 3 type = pres19
```

(Confirm the channel type)

```
>> calibration 3 datetime = 20130401120000, c0 = 0.2346, c1 = 120.9873, c2 = 2. 7356, c3 = 0.7
```

(Set the core coefficients)

```
>> calibration 3 datetime = 20130401120005, x0 = 9.983, x1 = 0.2003, x2 = 0.2943, x3 = 0.0721, x4 = 0.1049, x5 = 21.29
```

(Set the cross-channel correction coefficients)

```
>> calibration 3
<< calibration 3 type = pres19, datetime = 20130401120005, c0 = 0.2346, c1 = 120.9873, c2 = 2. 7356, c3 = 0.7, x0 = 9.983, x1 = 0.2003, x2 = 0.2943, x3 = 0.0721, x4 = 0.1049, x5 = 21.29, n0 = 2
```

(Request confirmation of everything)

7.3.4 Example 4: corr_cond - Conductivity corrections

Continuing with the RBR*concerto* C.T.D.pH logger, the conductivity reading from Channel-1 without corrections is given by a simple linear function:

$$Craw = c0 + c1*R$$

where R is the normalized voltage ratio from Channel-1 monitoring conductivity, **c0,c1** are the core coefficients of the linear equation, and Craw is the uncorrected conductivity output, reported in mS/cm for RBR marine instruments.

The equation which corrects the output for the effects of both temperature and pressure on the conductivity cell is:

$$Ccorr = \frac{Craw - Kc1*(T-Tcal)}{1 + Kc2*(T-Tcal) + Kc3*(Pcorr-Pcal)}$$

Casting the equation into the style used by the logger would give:

```
                Craw - x0*(value(n0)-x3)
Ccorr = ----------------------------------------
        1 + x1*(value(n0)-x3) + x2*(value(n1)-x4)
```

where

- Craw is the uncorrected conductivity, **c0** + **c1***R,*
- **x0 , x1 , x2** correspond directly to the constants 'Kc1', 'Kc2' and 'Kc3',
- **x3** is the calibration temperature 'Tcal' in °C,
- **x4** is the calibration pressure 'Pcal' in dbar,
- **n0** is the index of the temperature channel, in this example 2,
  value(**n0**) is the final output value of the temperature channel in °C,
- **n1** is the index of the pressure channel, in this example 3,
  value(**n1**) is the final output value of the pressure channel in dbar,
- Ccorr is the corrected output in mS/cm.

It is quite common to have a logger monitoring conductivity and temperature without a pressure channel, typically deployed at a known, constant depth. In this case, **n1** would be set to "**value** ", and so value(**n1**) would be substituted by a default value (see the "**settings pressure**" command).

Example commands:

```
>> calibration 1 type
<< calibration 1 type = cond06
```

(Confirm the channel type)

```
>> calibration 1 datetime = 20130401120000, c0 = 0.2346, c1 = 153.4873
```

(Set the core coefficients)

```
>> calibration 1 datetime = 20130401120005, x0 = 0.2003, x1 = 0.2943, x2 = 0.085, x3 =
15.028, x4 = 10.0025
```

(Set the cross-channel correction coefficients)

```
>> calibration 1

<< calibration 1 type = cond05, datetime = 20130401120005, c0 = 0.2346, c1 = 153.4873,
x0 = 0.2003, x1 = 0.2943, x2 = 0.085, x3 = 15.028, x4 = 10.0025, n0 = 2, n1 = 3
```

(Request confirmation of everything)

7.3.5 Example 5: corr_rinko - Correction of Rinko Dissolved Oxygen using Rinko Temperature sensor

> **RINKO formula revision**
>
> In November 2013, JFE Advantech introduced a revised version of the equation relating sensor voltage output to dissolved oxygen saturation. The revised equation is referred to as the (B) version; *this* example describes the older, original equation. For a description of the (B) version, refer to Example 11 (page 168).
>
> Sensors calibrated according to the (B) version of the equation *should* be labelled as such, so if the sensor has no label it probably uses this original equation. However, there is always the possibility that this label was not applied or is now missing; for example, older sensors returned to the manufacturer to be recalibrated will be processed using the (B) equation, and may not be re-labelled. The most reliable way to determine which equation applies is to check the most recent calibration certificate from the manufacturer.
>
> It is obviously important that the correct equation is used by the logger, which requires the correct channel type to be set in the logger's configuration (see the channel (page 78) command, and the Section Supported Channel Types (page 139)). The original equation described in this example requires channel type 'doxy02'. If you believe that the logger configuration is not correct for your sensor, please contact RBR Ltd for assistance.

Consider now an RBR*concerto* C.T.D.DO logger, where the DO (dissolved oxygen) channel is a Rinko-III sensor from JFE Advantech. This sensor also has its own temperature output, used to compensate the temperature dependence of the raw DO output, and to monitor this requires a fifth channel in the logger, Tr. See the section corr_rinkotemp - Correction of Rinko Dissolved Oxygen using logger Temperature sensor (page 161) for details of this temperature channel.

The equation for the temperature corrected output of the DO sensor, as provided by JFE Advantech, is as follows.

$$DOtcomp = \frac{A}{1 + D*(Tr-25)} + \frac{B}{(N-F)(1 + D(Tr-25)) + C + F}$$

where

A,B,C,D,F are coefficients provided by JFE Advantech,

N is the DO sensor output voltage in Volts,

Tr is the compensating temperature, also from the Rinko-III sensor.

The form of this equation used by the logger is:

```
                x0                              x1
DOtcomp = ---------------------- + ------------------------------------------
          1 + x3*(value(n0)-25)     (N-x5)(1 + x3(value(n0)-25)) + x2 + x5
```

where

- **x0** , **x1** , **x2** , **x3** , **x5** map directly and trivially to the coefficients A,B,C,D,F,
- value(**n0**) is the compensating temperature.

Note carefully that in this example the value of **n0** would be 5, using the temperature measured by the Rinko-III sensor itself. The temperature measured by the logger on Channel-2 is not suitable for direct use, because it does not have a time response which matches that of the Rinko-III DO sensor.

DOtcomp is then the input to a simple linear equation, which according to JFE Advantech can be further compensated for pressure effects as follows:

```
DOcorr = (G + H*DOtcomp) * (1 + E*P)
```

where

E,G,H are coefficients provided by JFE Advantech,

P is pressure in Mpa,

DOcorr is the fully corrected dissolved oxygen output in percentage saturation.

The G,H coefficients can be modified by the user to update the calibration of the sensor; the remaining coefficients should not need to be modified.

Casting the equation into the form used by the logger gives:

```
DOcorr = (c0 + c1*DOtcomp) * (1 + x4*value(n1))
```

where

- **c0**,**c1** are G, H,
- **x4** is E,

- **n1** is the index of the pressure channel, in this example 3
  value(**n1**) is the final output value of the pressure channel in dbar.

The logger correctly accounts for the fact that value(**n1**) is in dbar but the value of coefficient **x4** (E) is determined assuming the pressure to be in MPa.

Example commands:

```
>> calibration 4 type
<< calibration 4 type = doxy02
```

(Confirm the channel type)

```
>> calibration 4 datetime = 20130401120000, c0 = 0.346, c1 = 1.08873
```

(Set the 'user' coefficients)

```
>> calibration 4 datetime = 20130401120005, x0 = -41.7148, x1 = 25.425, x2 = -0.08097,
x3 = 0.0021, x4 = 4.5e-5, x5 = 0.0
```

(Set the coefficients provided by JFE Advantech)

```
>> calibration 4
<< calibration 4 type = doxy02, datetime = 20130401120005, c0 = 0.346, c1 = 1.08873, x0
= -41.7148, x1 = 25.425, x2 = -0.08097, x3 = 0.0021, x4 = 4.5e-5, x5 = 0.0, n0 = 5, n1 = 3
```

(Request confirmation of everything)

7.3.6 Example 6: corr_rinkoT - Correction of Rinko Dissolved Oxygen using logger Temperature sensor

RINKO formula revision

In November 2013, JFE Advantech introduced a revised version of the equation relating sensor voltage output to dissolved oxygen saturation. The revised equation is referred to as the (B) version; *this* example describes the older, original equation. For a description of the (B) version, refer to Example 12 (page 171).

Sensors calibrated according to the (B) version of the equation *should* be labelled as such, so if the sensor has no label it probably uses this original equation. However, there is always the possibility that this label was not applied or is now missing; for

example, older sensors returned to the manufacturer to be recalibrated will be processed using the (B) equation, and may not be re-labelled. The most reliable way to determine which equation applies is to check the most recent calibration certificate from the manufacturer.

It is obviously important that the correct equation is used by the logger, which requires the correct channel type to be set in the logger's configuration (see the channel (page 78) command, and the Section Supported Channel Types (page 139)). The original equation described in this example requires channel type 'doxy06'. If you believe that the logger configuration is not correct for your sensor, please contact RBR Ltd for assistance.

Consider an RBR*concerto* C.T.D.DO logger, where the DO channel is a Rinko-III dissolved oxygen sensor from JFE Advantech. For a number of reasons, it may be desirable to use the logger's measured temperature to compensate the temperature dependence of the raw DO output, rather than the built-in Rinko-III temperature sensor. This can be done, but requires some additional calculation, because as pointed out earlier (see the section corr_rinko - Correction of Rinko Dissolved Oxygen using Rinko Temperature sensor (page 159)), the time response of the logger's temperature sensor does not match that of the Rinko-III DO sensor.

There is no change to the primary equation for the temperature corrected output of the DO sensor:

$$DOtcomp = \frac{A}{1 + D*(Tr-25)} + \frac{B}{(N-F)(1 + D(Tr-25)) + C + F}$$

as before, but in this case Tr is a 'delayed' version of the monitored temperature, computed by the following simple filter operation:

$$Tr(n) = K*T + (1-K)*Tr(n-1)$$

where

- T is the monitored temperature,
- Tr(n) is the delayed temperature output,
- Tr(n-1) is the previous delayed temperature output,
- K is a term relating the measurement interval and the time constants of the sensors as follows

$$K = P / (TCdo - TCt)$$

where

- TCdo is the time constant of the Rinko-III Do sensor,
- TCt is the time constant of the logger's temperature sensor,
- P is the logger's measurement interval set by the "**sampling period**" command.

The time constants are specified in seconds using the additional auxiliary coefficients **x6** (TCdo) and **x7** (TCt); the logger correctly accounts for the fact that the measurement interval P is specified in milliseconds.

Often the logger's temperature sensor has a much faster response than the Rinko-III DO sensor, in which case it is acceptable to set TCt to zero. Also, in cases where the measurement interval P is long compared to the sensor time constants, the logger limits the computed value of K to 1, in which case the 'delayed' temperature Tr(n) follows the input temperature T exactly.

The form of the delayed temperature equation used by the logger becomes:

```
Tr(n)= K*value(n0) + (1-K)*Tr(n-1)
```

In the example discussed here, the value of **n0** is 2, and value(**n0**) is the temperature reported by the logger.

Example commands:

```
>> calibration 4 type
<< calibration 4 type = doxy06
```

(Confirm the channel type)

```
>> calibration 4 datetime = 20130401120000, c0 = 0.346, c1 = 1.08873
```

(Set the 'user' coefficients)

```
>> calibration 4 datetime = 20130401120005, x0 = -41.7148, x1 = 25.425, x2 = -0.08097,
x3 = 0.0021, x4 = 4.5e-5, x5 = 0.0, x6 = 4.2, x7 = 0.0
```

(Set the coefficients provided by JFE Advantech, and the time constants)

```
>> calibration 4
```

```
<< calibration 4 type = doxy02, datetime = 20130401120005, c0 = 0.346, c1 = 1.08873, x0 = -41.7148, x1 = 25.425, x2 = -0.08097, x3 = 0.0021, x4 = 4.5e-5, x5 = 0.0, x6 = 4.2, x7 = 0.0, n0 = 5, n1 = 3
```

(Request confirmation of everything)

7.3.7 Example 7: pss78 - derivation of Practical Salinity (1978)

Full, in-situ derivation of salinity requires that conductivity, temperature and pressure are all measured, so a simple RBR*concerto* C.T.D will be used as an example.

The equation relating salinity to the three underlying parameters is the Practical Salinity Scale of 1978, often referred to as PSS78: for further information refer to the section Practical Salinity of Seawater (page 182). The equation involves a rather fearsome looking series of polynomials combined in various ways: mercifully the coefficients are all empirically determined constants, and all values are embedded in the logger.

Salinity is a 'pure' derived parameter which has its own channel assigned to it, but there is no underlying measurement hardware for salinity itself; it simply uses the outputs of the conductivity, temperature and pressure channels. This makes its specification rather sparse: there are no coefficients in either of the '**c**' or '**x**' groups; all that is needed is to specify the indices in the '**n**' group.

Because hydrostatic pressure is used in the salinity equation, it also accommodates the presence of a channel to measure atmospheric pressure. In practice most loggers, like the RBR *concerto* C.T.D in this example, will not have an "atmospheric pressure" channel, so the '**n**' group index will be set to the text "value", and the logger will use the substitute value specified by the "**settings atmosphere**" command.

In our example:

- **n0** is the index of the temperature channel, 3 in this example,
- **n1** is the index of the pressure channel, 2 in this example,
- **n2** is the index of the conductivity channel, 1 in this example,
- **n3** is the index of the atmospheric pressure channel; not present in this example, so set to "value".

Example commands:

```
>> calibration 4 type

<< calibration 4 type = sal_00
```

(Confirm the channel type)

```
>> calibration 4
<< calibration 4 type = sal_00, datetime = 20130401120013, n0 = 3, n1 = 2, n2 = 1, n3 = value
```

(Request confirmation of everything)

It is not uncommon to monitor salinity using a logger with only conductivity and temperature (C. T) channels, deployed at a constant depth. In this case we might have:

```
>> calibration 4
<< calibration 4 type = sal_00, datetime = 20130401120013, n0 = 2, n1 = value, n2 = 1, n3 = value
```

(Request confirmation of everything)

7.3.8 Example 8: seapres - derivation of sea pressure from pressure

The sea pressure (also referred to as hydrostatic pressure) is simply the difference between pressure measured underwater and atmospheric pressure.

Using the example of an RBR*concerto* C.T.D, for this derived channel:

- **n0** is the index of the pressure channel, 3 in this case
- **n1** is the index of the atmospheric pressure channel; not present in this example, so set to "value".

Example commands:

```
>> calibration 4 type
<< calibration 4 type = pres08
```

(Confirm the channel type)

```
>> calibration 4
<< calibration 4 type = pres08, datetime = 20130401120013, n0 = 3, n1 = value
```

(Request confirmation of everything)

7.3.9 Example 9: depth - derivation of depth from pressure

This derived channel implements a simplified equation for water depth in meters, in which no account is taken of either geographical variations in the Earth's gravitational field, or the variation of water density with depth: both these quantities are treated as constants, although the water density can be changed using the "**settings density**" command.

```
Dm = (P – Patm)/( * g)
```

In the form used by the logger, using an RBR*concerto* C.T.D as an example, this becomes

```
Dm = (value(n0) – value(n1))/( * g)
```

where

- **n0** is the index of the pressure channel, 3 in this case,
- **n1** is the index of the atmospheric pressure channel; not present in this example, so set to "value".
- is the value set for the density of water using the "**settings density**" command,g is a fixed constant 0.980665, representing the standard value of acceleration due to gravity, in units which correctly account for pressures being measured in decibars,
- Dm is the calculated depth in meters.

Example commands:

```
>> calibration 4 type
<< calibration 4 type = dpth01
```

(Confirm the channel type)

```
>> calibration 4
<< calibration 4 type = dpth01, datetime = 20130401120013, n0 = 3, n1 = value
```

(Request confirmation of everything)

```
>> settings density
<< settings density = 1.026021
>> settings density = 1.0197
```

```
<< settings density = 1.0197
```

(Request, then change, the programmed water density)

7.3.10 Example 10: corr_metsmeth - Temperature correction of METS methane output

An RBR*concerto* C.T.D.METS has a METS methane sensor from Franatech in addition to the usual C, T and D channels. The METS sensor also has its own temperature output, used to compensate the temperature dependence of the raw methane output, and to monitor this requires a fifth channel in the logger, Tm. See the section corr_metstemp - Temperature measured by a METS (methane sensor) (page 150) for details of this temperature channel.

The equation for the temperature corrected concentration output of the methane sensor, Cm, as provided by Franatech, is as follows:

Cm = exp[c0*ln{ (c1 + c2*exp(-Vt/c3)) * (1/Vm - 1/(c4 + c5*exp(-Vt/c6))) }]

where

- **c0**...**c6** are coefficient values provided by Franatech,
- Vm is the voltage in Volts from the sensor's methane output,
- Vt is the voltage in Volts from the sensor's temperature output.

Franatech's calibration documentation may have no formal naming convention for the coefficients, in which case the values will be simply shown in an equation on the calibration sheet. The terms **c0**...**c6** are those used by the logger's calibration command to report or set the values: if updating, be careful to assign the correct values to each coefficient.

The sensor output voltage for methane concentration, Vm, is related to the logger's reported voltage ratio R by a simple linear equation, the coefficients of which (**x0**,**x1**) are determined by RBR Ltd at the factory.

```
Vm = x0 + x1*R
```

The sensor output voltage for temperature, Vt, is an intermediate variable in the calculation of temperature from the supporting channel, and can be back-calculated from that result:

```
Vt = (Tm-ct0)/ct1
```

where

- **ct0**, **ct1** are the primary coefficient values for the supporting temperature channel,
- Tm is the temperature output of that channel in °C, value(**n0**).

Example commands:

```
>> calibration 4 type
<< calibration 4 type = meth00
```

(Confirm the channel type)

```
>> calibration 4 datetime = 20130401120000, c0 = 1.269, c1 = 0.104, c2 = 3.268, c3 = 0.551, c4 = 0.830, c5 = 4.756, c6 = 1.432
```

(Set the coefficients provided by Franatech)

```
>> calibration 4 datetime = 20130401120005, x0 = 6.782656, x1 = -9.257345
```

(Set the secondary coefficients for the voltage conversion)

```
>> calibration 4
<< calibration 4 type = meth00, datetime = 20130401120005, c0 = 1.269, c1 = 0.104, c2 = 3.268, c3 = 0.551, c4 = 0.830, c5 = 4.756, c6 = 1.432, x0 = 6.782656, x1 = -9.257345, n0 = 5
```

(Request confirmation of everything for primary methane channel)

```
>> calibration 5 c0 c1
<< calibration 5 c0 = -5.65608, c1 = 16.80047
```

(Confirm main coefficients of temperature compensation channel)

7.3.11 Example 11: corr_rinkoB - Correction of Rinko Dissolved Oxygen using Rinko Temperature sensor

> **RINKO formula revision**
>
> In November 2013, JFE Advantech introduced a revised version of the equation relating sensor voltage output to dissolved oxygen saturation. The revised equation is referred to as the (B) version, and is described in *this* example. For a description of the older, original equation, refer to Example 5 (page 159).

Sensors calibrated according to the (B) version of the equation *should* be labelled as such, so if the sensor has no label it probably uses the original equation. However, there is always the possibility that this label was not applied or is now missing; for example, older sensors returned to the manufacturer to be recalibrated will be processed using the (B) equation, and may not be re-labelled. The most reliable way to determine which equation applies is to check the most recent calibration certificate from the manufacturer.

It is obviously important that the correct equation is used by the logger, which requires the correct channel type to be set in the logger's configuration (see the channel (page 78) command, and the Section Supported Channel Types (page 139)). The (B) version of the equation described in this example requires channel type 'doxy08'. If you believe that the logger configuration is not correct for your sensor, please contact RBR Ltd for assistance.

Consider an RBR*concerto* C.T.D.DO logger, where the DO (dissolved oxygen) channel is a Rinko-III sensor from JFE Advantech. This sensor also has its own temperature output, used to compensate the temperature dependence of the raw DO output, and to monitor this requires a fifth channel in the logger, Tr. See the section corr_rinkotemp - Correction of Rinko Dissolved Oxygen using logger Temperature sensor (page 161) for details of this temperature channel.

The equation for the temperature corrected output of the DO sensor, as provided by JFE Advantech, is as follows.

$$DOtcomp = \frac{A}{1 + D*(Tr-25) + F*(Tr-25)^2} + \frac{B}{N*(1 + D*(Tr-25) + F*(Tr-25)^2) + C}$$

where

A,B,C,D,F are coefficients provided by JFE Advantech,

N is the DO sensor output voltage in Volts,

Tr is the compensating temperature, also from the Rinko-III sensor.

The form of this equation used by the logger is:

$$DOtcomp = \frac{x0}{1 + x3*(value(n0)-25) + x5*(value(n0)-25)^2} + \frac{x1}{N*(1 + x3*(value(n0)-25) + x5*(value(n0)-25)^2) + x2}$$

where

- **x0** , **x1** , **x2** , **x3** , **x5** map directly and trivially to the coefficients A,B,C,D,F,
- value(**n0**) is the compensating temperature.

Note carefully that in this example the value of **n0** would be 5, using the temperature measured by the Rinko-III sensor itself. The temperature measured by the logger on Channel-2 is not suitable for direct use, because it does not have a time response which matches that of the Rinko-III DO sensor.

DOtcomp is then the input to a simple linear equation, which according to JFE Advantech can be further compensated for pressure effects as follows:

```
DOcorr = (G + H*DOtcomp) * (1 + E*P)
```

where

E,G,H are coefficients provided by JFE Advantech,

P is pressure in Mpa,

DOcorr is the fully corrected dissolved oxygen output in percentage saturation.

The G,H coefficients can be modified by the user to update the calibration of the sensor; the remaining coefficients should not need to be modified.

Casting the equation into the form used by the logger gives:

```
DOcorr = (c0 + c1*DOtcomp) * (1 + x4*value(n1))
```

where

- **c0**,**c1** are G, H,
- **x4** is E,
- **n1** is the index of the pressure channel, in this example 3
  value(**n1**) is the final output value of the pressure channel in dbar.

The logger correctly accounts for the fact that value(**n1**) is in dbar but the value of coefficient **x4** (E) is determined assuming the pressure to be in MPa.

Example commands:

```
>> calibration 4 type
<< calibration 4 type = doxy08
```

(Confirm the channel type)

```
>> calibration 4 datetime = 20130401120000, c0 = 0.346, c1 = 1.08873
```

(Set the 'user' coefficients)

```
>> calibration 4 datetime = 20130401120005, x0 = -41.7148, x1 = 25.425, x2 = -0.08097,
x3 = 0.0021, x4 = 4.5e-5, x5 = 0.0
```

(Set the coefficients provided by JFE Advantech)

```
>> calibration 4

<< calibration 4 type = doxy08, datetime = 20130401120005, c0 = 0.346, c1 = 1.08873, x0
= -41.7148, x1 = 25.425, x2 = -0.08097, x3 = 0.0021, x4 = 4.5e-5, x5 = 0.0, n0 = 5, n1 = 3
```

(Request confirmation of everything)

7.3.12 Example 12: corr_rinkoTB - Correction of Rinko Dissolved Oxygen using logger Temperature sensor

RINKO formula revision

In November 2013, JFE Advantech introduced a revised version of the equation relating sensor voltage output to dissolved oxygen saturation. The revised equation is referred to as the (B) version, and is described in *this* example. For a description of the older, original equation, refer to Example 6 (page 161).

Sensors calibrated according to the (B) version of the equation *should* be labelled as such, so if the sensor has no label it probably uses the original equation. However, there is always the possibility that this label was not applied or is now missing; for example, older sensors returned to the manufacturer to be recalibrated will be processed using the (B) equation, and may not be re-labelled. The most reliable way to determine which equation applies is to check the most recent calibration certificate from the manufacturer.

It is obviously important that the correct equation is used by the logger, which requires the correct channel type to be set in the logger's configuration (see the channel (page 78) command, and the Section Supported Channel Types (page 139)). The (B) version of the equation described in this example requires channel type 'doxy09'. If you believe that the logger configuration is not correct for your sensor, please contact RBR Ltd for assistance.

Consider an RBR*concerto* C.T.D.DO logger, where the DO channel is a Rinko-III dissolved oxygen sensor from JFE Advantech. For a number of reasons, it may be desirable to use the logger's measured temperature to compensate the temperature dependence of the raw DO output, rather than the built-in Rinko-III temperature sensor. This can be done, but requires some additional calculation, because as pointed out earlier (see the section corr_rinko - Correction of Rinko Dissolved Oxygen using Rinko Temperature sensor (page 159)), the time response of the logger's temperature sensor does not match that of the Rinko-III DO sensor.

There is no change to the primary equation for the temperature corrected output of the DO sensor:

$$DOtcomp = \frac{A}{1 + D*(Tr-25) + F*(Tr-25)^2} + \frac{B}{N*(1 + D*(Tr-25) + F*(Tr-25)^2) + C}$$

as before, but in this case Tr is a 'delayed' version of the monitored temperature, computed by the following simple filter operation:

$$Tr(n) = K*T + (1-K)*Tr(n-1)$$

where

- T is the monitored temperature,
- Tr(n) is the delayed temperature output,
- Tr(n-1) is the previous delayed temperature output,
- K is a term relating the measurement interval and the time constants of the sensors as follows

$$K = P / (TCdo - TCt)$$

where

- TCdo is the time constant of the Rinko-III Do sensor,
- TCt is the time constant of the logger's temperature sensor,
- P is the logger's measurement interval set by the "**sampling period**" command.

The time constants are specified in seconds using the additional auxiliary coefficients **x6** (TCdo) and **x7** (TCt); the logger correctly accounts for the fact that the measurement interval P is specified in milliseconds.

Often the logger's temperature sensor has a much faster response than the Rinko-III DO sensor, in which case it is acceptable to set TCt to zero. Also, in cases where the measurement interval P is long compared to the sensor time constants, the logger limits the computed value of K to 1, in which case the 'delayed' temperature Tr(n) follows the input temperature T exactly.

The form of the delayed temperature equation used by the logger becomes:

```
Tr(n)= K*value(n0) + (1-K)*Tr(n-1)
```

In the example discussed here, the value of **n0** is 2, and value(**n0**) is the temperature reported by the logger.

Example commands:

```
>> calibration 4 type
<< calibration 4 type = doxy09
```

(Confirm the channel type)

```
>> calibration 4 datetime = 20130401120000, c0 = 0.346, c1 = 1.08873
```

(Set the 'user' coefficients)

```
>> calibration 4 datetime = 20130401120005, x0 = -41.7148, x1 = 25.425, x2 = -0.08097,
x3 = 0.0021, x4 = 4.5e-5, x5 = 0.0, x6 = 4.2, x7 = 0.0
```

(Set the coefficients provided by JFE Advantech, and the time constants)

```
>> calibration 4
<< calibration 4 type = doxy09, datetime = 20130401120005, c0 = 0.346, c1 = 1.08873, x0
= -41.7148, x1 = 25.425, x2 = -0.08097, x3 = 0.0021, x4 = 4.5e-5, x5 = 0.0, x6 = 4.2, x7 =
0.0, n0 = 5, n1 = 3
```

(Request confirmation of everything)

7.3.13 Example 13: deri_sos, speed of sound

Full, in-situ derivation of speed of sound requires salinity, hydrostatic pressure and temperature, so a simple RBR*concerto* C.T.D will be used as an example.

The equation used in the logger relating speed of sound to the three underlying parameters is the Chen and Millero equation reviewed by Wong and Zhu, often referred to as UNESCO equation. For further information about this equation, please refer to the paper: G.S.K. Wong and S Zhu, Speed of sound in seawater as a function of salinity, temperature and pressure (1995) J. Acoust. Soc. Am. 97(3) pp 1732-1736.

The equation involves a rather fearsome looking series of polynomials combined in various ways: mercifully the coefficients are all empirically determined constants, and all values are embedded in the logger.

Speed of sound is a 'pure' derived parameter which has its own channel assigned to it, but there is no underlying measurement hardware for speed of sound itself; it simply uses the outputs of the salinity, temperature and hydrostatic pressure channels. This makes its specification rather sparse: there are no coefficients in either of the '**c**' or '**x**' groups; all that is needed is to specify the indices in the '**n**' group.

In our example:

- **n0** is the index of the temperature channel, 3 in this example,
- **n1** is the index of the hydrostatic pressure channel, 4 in this example,
- **n2** is the index of the salinity channel, 6 in this example,

Example commands:

```
>> calibration 7 type
<< calibration 7 type = sos_00
```

(Confirm the channel type)

```
>> calibration 7
<< calibration 7 type = sos_00, datetime = 20130401120013, n0 = 3, n1 = 4, n2 = 6
```

(Request confirmation of everything)

7.3.14 Example 14: deri_speccond, specific conductivity

This equation permits conductivity readings taken in different environments to be compared by correcting them to a standard environment at 25°C. Specific conductivity is usually of greater interest in fresh water applications, and is by convention always reported in S/cm, although the parameter does apply to salt water as well. The equation which corrects for temperature to derive specific conductivity from standard conductivity is given below. It is associated with the **scon00** derived channel type.

```
                      K0
C25 = Ccorr * ---------------------
              1 + K1*(T-25)
```

where

- **Ccorr** is the standard conductivity reading (already compensated for temperature dependence of the measurement circuit as described in Example 4 (page 157)),
- **T** is the temperature used for correction, in °C,
- **K0** is a units correction factor, and
- **K1** is a temperature coefficient.

In the calibration settings for the **scon00** derived channel type, the channel cross-reference index for Ccorr is given by **n0**, and for T by **n1**.

K0 has a value of 1 if the Ccorr channel is in S/cm, or a value of 1000 if Ccorr is in mS/cm. The logger can deduce this from the units of the Ccorr channel; an explicit coefficient is not needed.

K1 depends on the ionic composition of the water being monitored, and typically has a value in the range 0.0191 to 0.0214. The lower end of this range is suitable for KCl solutions, the higher end for NaCl solutions. The value used by the logger can be queried and modified via the settings (page 83) command, using the **speccondtempco** parameter. If this parameter is never explicitly set, the default value used is 0.0191.

7.3.15 Example 15: deri_bprpres and deri_bprtemp, BPR channels

Loggers with BPR channels interface a Paroscientific Inc. transducer. The logger measure precisely the output frequencies from the transducer. Those transducers generally outputs two signals, one for pressure and one for temperature.

There are two channels for the periods measured (peri_00 and peri_01). But in order to convert them in a meaningful pressure and temperature, the logger provides two channels which implement the calibration equation from Paroscientific Inc. They rely on two type of equations: deri_bprpres (pressure in dbar) and deri_bprtemp (temperature in °C).

The following applies to loggers with channels **bpr_08** and **bpr_09** types (loggers manufactured since november 2015). For former types of BPR channels, pleaser refer to the bpr (page 86) command.

deri_bprpres equation

```
Pressure (dbar) = ( C * (1 - (T0/T)^2)) * (1 - D(1 - (T0/T)^2)) ) * 0.689475728
```

```
T = value(n0) * 1e-6
X = value(n1) * 1e-6
U = X - x0
C = x1 + x2*U + x3*U^2
D = x4 + x5*U
T0 = x6 + x7*U + x8*U^2 + x9*U^3 + x10*U^4
```

- The coefficients **x0 ... x10** are provided by Paroscientic Inc. and obeys to the following bijection:

Paroscientific Inc. coefficient	U0	C1	C2	C3	D1	D2	T1	T2	T3	T4	T5
deri_bprpres coefficient	**x0**	**x1**	**x2**	**x3**	**x4**	**x5**	**x6**	**x7**	**x8**	**x9**	**x10**

- **n0** is the index of the pressure period channel, in this example 1
- value(**n0**) is the final output value of the pressure period channel in picoseconds.
- **n1** is the index of the temperature period channel, in this example 2
- value(**n1**) is the final output value of the temperature period channel in picoseconds.

deri_bprtemp equation

```
Temperature = x1*U + x2*U^2 + x3*U^3
X = value(n0) * 1e-6
U = X - x0
```

- The coefficients **x0 ... x3** are provided by Paroscientic Inc. and obeys to the following bijection:

Paroscientific Inc. coefficient	U0	Y1	Y2	Y3
deri_bprtemp coefficient	**x0**	**x1**	**x2**	**x3**

- **n0** is the index of the temperature period channel, in this example 2
- value(**n0**) is the final output value of the temperature period channel in picoseconds.

Example commands

```
>> calibration 3 type
<< calibration 3 type = bpr_08

>> calibration 4 type
<< calibration 4 type = bpr_09
```

(Confirm the channel type)

```
>> calibration 3 datetime= 20151123120721, x0 = 5.8310298E+00, x1 = -24.514029E+03,
x2 = -573.64117E+00, x3 = 76.129281E+03, x4 = 35.688001E-03, x5 = 0.0000000E+00,
x6 = 30.413169E+00, x7 = 664.14898E-03

<< calibration 3 datetime = 20151123120721, x0 = 5.8310300e+000, x1 = -24.514030
e+003, x2 = -573.64115e+000, x3 = 76.129280e+003, x4 = 35.688000e-003, x5 =
0.0000000e+000, x6 = 30.413170e+000, x7 = 664.14899e-003
>> calibration 3 x8 = 58.803409E+00, x9 = 180.91160E+00, DATETIME =
20151123120721, x10 = 0.0000000E+00
<< calibration 3 datetime = 20151123120721, x8 = 58.803408e+000, x9 = 180.91160
e+000, x10 = 0.0000000e+000
```

(Set the pressure coefficients provided by Paroscientific Inc.)

```
>> calibration 4 datetime = 20151123120722, x0 = 5.8310298E+00, x1 = -3.8981210E+03,
x2 = -10.493120E+03, x3 = 0.0000000E+00
<< calibration 4 datetime = 20151123120722, x0 = 5.8310300e+000, x1 = -3.8981210
e+003, x2 = -10.493120e+003, x3 = 0.0000000e+000
```

(Set the temperature coefficients provided by Paroscientific Inc.)

```
>> calibration 3
<< calibration 3 type = bpr_08, datetime = 20151123120721, x0 = 5.8310300e+000, x1 =
-24.514030e+003, x2 = -573.64115e+000, x3 = 76.129280e+003, x4 = 35.688000e-003,
x5 = 0.0000000e+000, x6 = 30.413170e+000, x7 = 664.14899e-003, x8 = 58.803408
e+000, x9 = 180.91160e+000, x10 = 0.0000000e+000, n0 = 1, n1 = 2

>> calibration 4
<< calibration 4 type = bpr_09, datetime = 20151123120722, x0 = 5.8310300e+000, x1 =
-3.8981210e+003, x2 = -10.493120e+003, x3 = 0.0000000e+000, n0 = 2
```

(Request confirmation of everything for pressure and temperature coefficients)

7.3.16 Example 16: distancefromechotiming Distance from echo timing

An RBR*concerto* Alti has a RBRalti sensor which outputs distance and echo timing. Only distance is reported by the RBR*concerto.*

Full, in-situ derivation of distance from echo timing requires an average speed of sound over the path used by the sound.

Distance is a 'pure' derived parameter which has its own channel assigned to it, but there is no underlying measurement hardware for distance itself; it simply uses the outputs of the echo timing channel and the average speed of sound setting. This makes its specification rather sparse: there are no coefficients in either of the '**c**' or '**x**' groups; all that is needed is to specify the indices in the '**n**' group.

In our example:

- **n0** is the index of the echo timing channel, **2** in this example,
- **n1** is the index of the average speed of sound channel, **value** in this example, ie the default setting.

The distance is calculated as follow:

```
Distance = Te * ASS / (1000 * 2.0)
```

where

- Te is the echo timing (round trip) in milliseconds, value(**n0**).
- ASS is the average sound of speed, value(**n1**) (refer to "settings avgsoundspeed (page 83)").

Example commands:

```
>> calibration 1 type
<< calibration 1 type = alti00
```

(Confirm the channel type)

```
>> calibration 1

<< calibration 1 type = alti00, datetime = 20130401120013, n0 = 2, n1 = value
```

(Request confirmation of everything)

7.3.17 Example 17: corr_o2conc_garcia, O2 concentration compensated for salinity and pressure

Consider an RBR*concerto* C.T.D.DO logger, where the DO channel is a RBR*optode*. The RBR *optode* transfers both the foil temperature and the dissolved oxygen concentration (not compensated for salinity). The RBR*concerto* calculates the concentration compensated for salinity and pressure first:

```
O2corr = (C0 + C1 * O2unc) * exp((S)*(Gb0+Gb1*Ts+Gb2*Ts^2+Gb3*Ts^3)+Gc0*(S^2)) *
( 1 + (Pressure-Atm. pressure)*C2 )
```

where

- O2corr is the corrected O2 concentration, compensated for salinity and pressure.
- O2unc is the uncompensated O2 concentration returned by the RBR*optode.*
- C0 and C1 are corrections and scaling factors for the uncompensated O2 concentration.
- C2 is a correction factor for pressure.
- S is the salinity in PSU

and

```
Ts = ln((298.15 - T)/(273.15 + T))
```

where T is the water temperature (in °C), and

```
Gb0=-6.24097e-3
Gb1=-6.93498e-3
Gb2=-6.90358e-3
Gb3=-4.29155e-3
Gc0=-3.11680e-7
```

which correspond to Garcia and Gordon coefficients.

The corresponding logger coefficients are:

- **c0** is C0
- **c1** is C1
- **c2** is C2
- **n0** is the index of the water temperature channel
- **n1** is the index of the salinity channel
- **n2** is the index of the pressure channel

- **n3** is the index of atm. pressure channel (set to **value** in order to use settings atmosphere (page 83))

> One might change c0 and c1 in order to perform a two point calibration to the RBR *optode*. Please note that those coefficients are used and stored only by the logger, not the RBR*optode*.

Example commands:

```
>> calibration 4 type
<< calibration 4 type = doxy23
```

(Confirm the channel type)

```
>> calibration 4
<< calibration 4 type = doxy23, datetime = 20160401120005, c0 = 0,c1 = 1, c2 = 3.25e-5,
n0 = 5, n1 = 8, n2 = 3, n3 = value
```

(Request confirmation of everything)

7.3.18 Example 18: deri_o2sat_garcia, Derived O2 saturation from concentration

Consider an RBR*concerto* C.T.D.DO logger, where the DO channel is a RBRoptode. The RBR *optode* transfers both the foil temperature and the dissolved oxygen concentration (not compensated for salinity). The RBR*concerto* calculates the concentration compensated for salinity first, then the air saturation (in %) via the following equation:

$$\text{Air saturation} = 2.24145 * \frac{(\text{Patm-Air vapor pressure})*Cc}{((10.1325-\text{Air vapor pressure})*\text{Solubility})}$$

Where Cc is the concentration in uMol/L, compensated for salinity.

Solubility is calculated via Gordon and Garcia as:

$$\text{Solubility} = \exp((Ga0+Ga1*Ts+Ga2*Ts^2+Ga3*Ts^3+Ga4*Ts^4+Ga5*Ts^5)+S*(Gb0+Gb1*Ts+Gb2*Ts^2+Gb3*Ts^3)+Gc0*S^2)$$

where S is the salinity in PSU and Ts is defined as:

```
Ts = ln((298.15 - T)/(273.15 + T))
```

with T being the water temperature in °C and:

```
Ga0=2.00856
Ga1=3.22400
Ga2=3.99063
Ga3=4.80299
Ga4=9.78188e-1
Ga5=1.71069
Gb0=-6.24097e-3
Gb1=-6.93498e-3
Gb2=-6.90358e-3
Gb3=-4.29155e-3
Gc0=-3.11680e-7
```

Air vapor pressure (in dbar) is calculated via :

```
Air vapor pressure = exp(52.57-6690.9/(T+273.15)-4.6818*ln(T+273.15))/100
```

The **n**-coefficients in the logger equation correspond to the following:

- **n0**, the index of the concentration channel, already compensated for salinity.
- **n1**, the index of the water temperature channel.
- **n2**, the index of the salinity channel.
- **n3**, the index of the atmospheric pressure channel. This last one is generally set to **value** , to use the default value provided by the logger; see **settings atmosphere** (page 83).

Example commands:

```
>> calibration 9 type
<< calibration 9 type = doxy22
```

(Confirm the channel type)

```
>> calibration 9
<< calibration 9 type = doxy22, datetime = 20130401120000, n0 = 4, n1 = 5, n2 = 8, n3 = value
```

(Request confirmation of everything)

7.4 Supporting Material

7.4.1 Practical Salinity of Seawater

Since it is not possible to directly measure the absolute salinity of seawater (the ratio of the mass of dissolved material to the mass of seawater), it is necessary to work in terms of practical salinity, which can be determined from measurable properties of seawater.

This is defined in "Algorithms for computation of fundamental properties of seawater", by N.P. Fotonoff and R.C.Millard Jr.:

The practical salinity, symbol S, of a sample of sea water, is defined in terms of the ratio K of the electrical conductivity of a sea water sample of 15°C and the pressure of one standard atmosphere, to that of a potassium chloride (KCl) solution, in which the mass fraction of KCl is 0.0324356, at the same temperature and pressure. The K value exactly equal to one corresponds, by definition, to a practical salinity equal to 35.

The practical salinity of seawater can be calculated from three measurable parameters: electrical conductivity, temperature, and pressure. Each of the three parameters is necessary for the salinity calculation since the electrical conductivity of seawater changes with temperature and pressure. Electrical conductivity of seawater is dependent upon the number of dissolved ions per volume (salinity), as well as the mobility of those ions (affected by temperature and pressure). The accuracy of the salinity 'measurement' depends on the accuracy to which the three principal parameters can be measured.

The Practical Salinity Scale of 1978, endorsed by UNESCO/IAPSO, is currently the world standard for salinity calculation. It is used by all RBR CTD instruments and software for the calculation of seawater salinity, using the equations given below; these are taken from "IEEE Journal of Oceanic Engineering", Vol. OE-5, No. 1, January 1980, page 14. Practical salinity, S, is given by:

```
S = a0 + a1*RT^0.5 + a2*RT^1.0 + a3*RT^1.5 + a4*RT^2.0 + a5*RT^2.5 + S
```

where S is a temperature correction term given by

```
          (T-15)
S = ----------------- * Fn(RT)
      1 + 0.0162*(T-15)
```

where Fn(RT) is the polynomial function

```
Fn(RT) = b0 + b1*RT^0.5 + b2*RT^1.0 + b3*RT^1.5 + b4*RT^2.0 + b5*RT^2.5
```

and T is the in-situ temperature according to the International Temperature Scale of 1968 (ITS-68). All RBR loggers and software use the more recent ITS-90 scale, but make the simple conversion to ITS-68 for salinity calculations.

RT is a term representing a ratio of conductivities, with further corrections applied for temperature and pressure:

```
RT = R / (Rp * rT)
```

R is the ratio of the conductivity of the sample of seawater (measured by the logger) to the conductivity of standard seawater at S = 35, T = 15ºC, and P = 0: Conductivity(35,15,0) = 42.914mS/cm.

```
      Conductivity(S,T,P)
R = ------------------------
      Conductivity(35,15,0)
```

Rp and rT are correction terms to adjust for in-situ pressure and temperature respectively:

```
                e1*P + e2*P^2 + e3*P^3
Rp = 1 + ---------------------------------
          1 + d1*T + d2*T^2 + (d3 + d4*T)*R
```

```
rT = c0 + c1*T + c2*T^2 + c3*T^3 + c4*T^4
```

where P is the in-situ hydrostatic pressure measured in bars (RBR loggers and software account for the conversion from decibars).

The table below gives all the coefficients required in all the above equations. These values have been empirically determined, and are fixed: they do not need to be programmed into a data logger in any way by end users.

Table 1. Coefficients for the PSS78 equations

	a	b	c	d	e
0	0.0080	0.0005	0.6766097		
1	-0.1692	-0.0056	2.00564e-2	3.426e-2	2.070e-4
2	25.3851	-0.0066	1.104259e-4	4.464e-4	-6.370e-8

	a	b	c	d	e
3	14.0941	-0.0375	-6.968e-7	0.4215	3.989e-12
4	-7.0261	0.0636	1.0031e-9	-3.107e-3	
5	2.7081	-0.0144			

8 Error messages

This is a current, but partial, list of error messages which the logger can produce. There are some messages which can not appear under normal operating conditions relevant to OEM users, and for the sake of simplicity these have not been included.

Each message begins with *Ennnn*, where *nnnn* is a 4-digit decimal number, padded with leading zeroes if necessary. The number allows host software to interpret the error code as desired if the rather terse messages from the logger are unsuitable for any reason.

Note that some messages may be followed by variable elements not shown here.

8.1 List of error messages

E0101	command parser busy
E0102	invalid command '<unknown-command-name>'
E0103	protected command, use 'permit <command>'
E0104	feature not yet implemented
E0105	command prohibited while logging
E0107	expected argument missing
E0108	invalid argument to command: '<invalid-argument>'
E0109	feature not available
E0110	buffer full
E0111	command failed
E0112	expected data missing
E0113	invalid data
E0114	feature not supported by hardware
E0300	unknown error
E0301	memory erase not completed
E0400	unknown error
E0401	estimated memory usage exceeds capacity
E0402	memory not empty, erase first

E0403	end time must be after start time
E0404	end time must be after current time
E0405	failed to enable for logging
E0406	not logging, stop = <logger-status>
E0408	logging already active
E0409	uncleared error, use 'errorlog'
E0410	no sampling channels active
E0411	period not valid for selected mode
E0412	burst parameters inconsistent
E0413	period too short for serial streaming
E0414	thresholding interval not valid
E0415	more than one gating condition is enabled
E0416	wrong regimes settings
E0417	no gating allowed with regimes mode
E0418	cast detection needs a pressure/depth channel
E0419	calibration coefficients are missing
E0420	required channel is turned off; <channel-index>
E0421	raw output format not allowed
E0422	AUX1 not available in current serial mode
E0423	wrong ddsampling settings

E0500	unknown error
E0501	item is not configured
E0502	configuration failed
E0503	all available channels assigned
E0504	address already in use
E0505	no channels configured
E0506	can not create calibration entry
E0507	can not delete calibration entry
E0600	unknown error
E0601	no calibration for channel '<channel-index>'