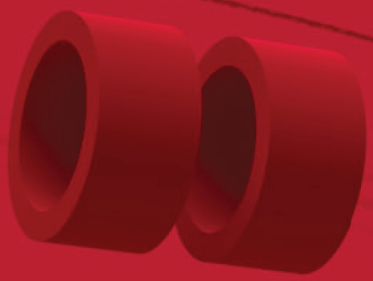
RBR

COMMAND REFERENCE

Generation 3

L3 version



rbr-global.com

Table of Contents

1 Introduction . . . 7

1.1 Document version history . . . 7

1.2 Formatting . . . 8

1.3 Security . . . 8

1.4 Command processing and timeouts . . . 9

1.4.1 Command entry . . . 9

1.4.2 Timeouts, output blanking and power saving . . . 10

1.4.3 Parsing logger responses . . . 11

1.4.4 Parameter modification . . . 12

2 Quick start . . . 13

2.1 General overview . . . 13

2.1.1 Channels . . . 13

2.1.2 Acquiring samples . . . 13

2.2 Enabling continuous sampling . . . 13

2.3 Enabling wave sampling . . . 14

2.4 Serial streaming from serial port . . . 14

2.4.1 Setting the correct baud rate . . . 14

2.4.2 Enabling the serial streaming . . . 14

2.5 Download stored data . . . 15

2.6 Downloading an EasyParse dataset . . . 16

2.7 Integrating with a profiling float . . . 17

2.7.1 Introduction . . . 17

2.7.2 Buoyancy control . . . 18

2.7.3 Setup for ascent, enable logging . . . 18

2.7.4 End of ascent, disable logging and download data . . . 20

2.7.5 More details on the calculation . . . 21

2.7.6 Available output channels . . . 21

2.7.7 Post-processing onboard . . . 21

2.7.8 Providing platform details to end-users . . . 23

2.7.9 Sensor drift monitoring at surface . . . 23

2.7.10 Energy tracking . . . 23

2.8 High resolution BPR and frequency counters for cabled ocean observatories . . . 23

2.8.1 Introduction . . . 23

2.8.2 Deploying a frequency counter/BPR logger running continuously at 1Hz and streaming over serial 23
2.8.3 BPR channels 24
2.8.4 Operating an RBRquartz³ BPR|zero instrument 25
2.9 Migrate from L2 to L3 platform 26
2.9.1 Introduction 26
2.9.2 Identifying the L3 platform 26
2.9.3 Deprecated commands 26
2.9.4 Removed commands 27
2.9.5 Improved commands and new parameters 28
2.9.6 Removed parameters 32
2.9.7 Syntactical changes 32
2.10 Tips for system integrators 32
2.10.1 Default deployment start and end time 32
2.10.2 Sampling rates 32
2.10.3 Future proofing development 33
2.10.4 Power management and power cycling behaviour 33
2.10.5 Memory format 34
2.10.6 Error handling 34
2.10.7 Electronic Static Discharge 35
3 Commands 36
3.1 Time and schedule 36
3.1.1 pause 36
3.1.2 clock 37
3.1.3 deployment 38
3.1.4 resume 39
3.1.5 pauseresume 40
3.1.6 sampling 41
3.2 Gated sampling 45
3.2.1 thresholding 45
3.2.2 twistactivation 47
3.3 Vehicle support 48
3.3.1 regimes 48
3.3.2 regime 49
3.3.3 ddsampling 52
3.4 Deployments 55

3.4.1 verify . . . 55
3.4.2 enable . . . 57
3.4.3 disable . . . 59
3.4.4 simulation . . . 60
3.5 Memory and data retrieval . . . 62
3.5.1 meminfo . . . 62
3.5.2 memclear . . . 64
3.5.3 memformat . . . 64
3.5.4 readdata . . . 66
3.5.5 postprocessing . . . 67
3.5.6 postprocessing_regimes . . . 70
3.5.7 postprocessing_regime . . . 71
3.6 Configuration information and calibration . . . 72
3.6.1 channels . . . 72
3.6.2 channel . . . 73
3.6.3 settings . . . 76
3.6.4 calibration . . . 78
3.6.5 sensor . . . 80
3.6.6 valve . . . 82
3.6.7 valvesegments . . . 85
3.6.8 valvesegment . . . 86
3.6.9 uvled . . . 89
3.7 Communications . . . 92
3.7.1 link . . . 92
3.7.2 serial . . . 93
3.7.3 sleep . . . 94
3.7.4 wifi . . . 95
3.8 Other Information . . . 97
3.8.1 id . . . 97
3.8.2 help . . . 97
3.8.3 hwrev . . . 98
3.8.4 power . . . 99
3.8.5 powerinternal . . . 99
3.8.6 powerexternal . . . 100
3.8.7 info . . . 101

3.8.8 getall . . . 102
3.9 Data sample . . . 104
3.9.1 fetch . . . 104
3.10 Security and Interaction . . . 105
3.10.1 permit . . . 105
3.10.2 prompt . . . 106
3.10.3 confirmation . . . 107
3.10.4 reboot . . . 108
3.11 Realtime data . . . 109
3.11.1 streamserial . . . 109
3.11.2 outputformat . . . 111
3.11.3 streamusb . . . 114
4 Format of stored data . . . 116
4.1 Overview . . . 116
4.1.1 Standard format . . . 116
4.1.2 EasyParse format . . . 116
4.2 EasyParse "calbin00" format . . . 117
4.2.1 EasyParse format events markers . . . 117
4.2.2 Sample data EasyParse format . . . 120
4.3 Standard "rawbin00" format . . . 122
4.3.1 Deployment header . . . 122
4.3.2 Sample data standard format . . . 137
4.3.3 Standard format events markers . . . 140
4.4 Profile detection events generation . . . 144
5 Supported channel types . . . 146
6 Calibration equations and cross-channel dependencies . . . 156
6.1 Core equations . . . 156
6.1.1 lin, or linear . . . 156
6.1.2 qad, or quadratic . . . 156
6.1.3 cub, or cubic . . . 156
6.1.4 tmp, or temperature . . . 156
6.2 Specialized equations . . . 156
6.2.1 corr_rinkotemp - temperature measured by a Rinko DO sensor . . . 156
6.2.2 corr_metstemp - temperature measured by a METS (methane sensor) . . . 157
6.2.3 optic2 - optical parameters measured by a Satlantic OCR sensor . . . 158

6.3 Dependent equations . . . 159
6.3.1 Example 1: corr_pH - simple temperature correction of pH . . . 159
6.3.2 Example 2: corr_pH - pH correction without temperature . . . 160
6.3.3 Example 3: corr_pres2 - temperature correction of pressure . . . 161
6.3.4 Example 4: corr_cond - conductivity corrections . . . 162
6.3.5 Example 5: corr_rinko - correction of Rinko dissolved oxygen using Rinko temperature sensor . . . 164
6.3.6 Example 6: corr_rinkoT - correction of Rinko dissolved oxygen using logger temperature sensor . . . 166
6.3.7 Example 7: pss78 - derivation of practical salinity (1978) . . . 167
6.3.8 Example 8: seapres - derivation of sea pressure from pressure . . . 168
6.3.9 Example 9: depth - derivation of depth from pressure . . . 169
6.3.10 Example 10: corr_metsmeth - temperature correction of METS methane output . . . 169
6.3.11 Example 11: corr_rinkoB - correction of Rinko dissolved oxygen using Rinko temperature sensor . . . 171
6.3.12 Example 12: corr_rinkoTB - correction of Rinko dissolved oxygen using logger temperature sensor . . . 173
6.3.13 Example 13: deri_sos, speed of sound . . . 175
6.3.14 Example 14: deri_speccond, specific conductivity . . . 176
6.3.15 Example 15: deri_bprpres and deri_bprtemp, BPR channels . . . 177
6.3.16 Example 16: distancefromechotiming distance from echo timing . . . 178
6.3.17 Example 17: corr_o2conc_garcia, O2 concentration compensated for salinity and pressure . . . 179
6.3.18 Example 18: deri_o2sat_garcia, derived O2 saturation from concentration . . . 180
6.3.19 Example 19: corr_cond1 - conductivity corrections for deep CT cell . . . 181
6.3.20 Example 20: corr_cond2 - conductivity corrections for CT cell . . . 183
6.3.21 Example 21: corr_cond3 - conductivity corrections for RBRLegato and 6000dbar C and CT cell . . . 184
6.3.22 Example 22: corr_pres5 - temperature correction of pressure . . . 185
6.3.23 Example 23: corr_irr - irradiance . . . 186
6.3.24 Example 24: corr_irr2 - generic irradiance and PAR . . . 187
6.3.25 Example 25: deri_dyncorrT and deri_dyncorrS dynamic correction channels . . . 188
6.3.26 Example 26: corr_cond4 - conductivity corrections for RBRLegato and 6000dbar C and CT cell . . . 191
6.4 Supporting material . . . 192
6.4.1 Practical salinity of seawater . . . 192
7 Error messages . . . 194
7.1 List of error and warning messages . . . 194

1 Introduction

This document applies *only* to RBR hardware that responds to the **id** command with a **fwtype** parameter value of 104.

1.1 Document version history

Auth.	Date	Rev.	Notes
GmJ/JmL/SC/BK/CJ/PF/XY	2018-2021	A-K	Various
JmL/XY/GmJ	4 December 2022	L	Updated **Integrating with a profiling float** for derived channels with dynamic correction. Added description of **valvesegments** and **valvesegment** commands. Added description of **scheduletype** parameter to **valve** command, supporting segmented schedules. Improved description of **twistactivation** **state**. Added **turb12**, **turb13**, **turb14**, **fluo43**, **fluo44**, **fluo47**, **temp38**, **sal_01** and **temp40** to **Supported Channel Types**. Added external battery type **fermata_nimh** to description of **powerexternal** command. Updated description of device control flags in the Deployment Header, advanced header version to 2.003. Updated description of event markers in Standard format data. Formatting correction in **sampling** command description. Updated **postprocessing** command. ***Internal revision, L3:*** *Added command #**bsl**.* *Documented secured options and behaviour for the **meminfo** command.* *Documented the segmented schedule table for valves in the Deployment Header.*

Auth.	Date	Rev.	Notes
GmJ	17 January 2024	M	Updated coefficient values in Gordon & Garcia equations, Example-17 and Example-18. Added channel types **turb19**, **turb20**, **temp42** to support Tu-25 turbidity sensor. Updated descriptions of **simulation** command and the Deployment Header to include a list of simulated **channels**. Corrected documentation of the **corr_metsmeth** equation in Example 10.
	20 October 2025	N	Added channel type **turb30**. Added channel types **ph__05, ph__06, ph__07, orp_03, orp_04, orp_05**.
JL/GmJ	06 January 2026	O	Add Supported channel types **cond26**, **cond28**. Added Example 26 for **corr_cond4** equation. Corrections to examples for other **corr_cond***X* equations.
VP	14 September 2026	P	Migrated. Reformatted and reviewed; no technical changes.

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. Reading parameters is always available.

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 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 cannot 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

Start and end of a command

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:

```
>> <CR><LF>
<< Ready:
>> <LF><CR>
<< Ready:
>> <CR><CR>
<< Ready: Ready:
>> <LF><LF>
<< Ready: Ready:
```

In the first two cases, the second character is considered redundant and is discarded; only one **Ready:** prompt is sent. For the last two cases, the second character is treated as a second empty command, so it also provokes the logger's prompt, and a total of two prompts are sent (see also the **prompt** command).

Upper case and lower case

In general, the logger is not sensitive to the case of the input; for example, **ID**, **Id**, **iD**, and **id** are all acceptable forms for the **id** command. Any exceptions to this rule are highlighted when necessary. However, when handling logger responses, do not assume that the case of the output will match the case of the input: see also Parsing logger responses.

Common error messages

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 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 will be returned by this character, but the logger will start to wake up.
% [10ms pause]
% The logger completes its wake-up procedure.
>> id<CR>
% The id command is a useful initial command as it replies with confirmation of the
instrument connection.
<< id model = RBRduo3, version = 1.000, serial = 050050, fwtype = 104
% 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. This timeout can be changed; refer to the description of the **inputtimeout** parameter for the **settings** 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, futureparameter = 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 (e.g., 21.7325), parsing this number without assuming anything about how many digits there are is more robust.

- When parsing streamed output or fetched values, it is good practice to assume some channels might report error code (see **outputformat** and **fetch**). If a channel reports an error code, other channels might still be valid.
- 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"). 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.

Some commands accept and respond with date/times which look like very large integers, but which have an implicit special format. For example, 20170401120000 represents noon on 1st April 2017. These cases are usually clear from the context.

1.4.4 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.

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 General overview

2.1.1 Channels

The RBR L3 platform based instruments allow users to measure a wide range of physical properties of the water. Temperature, pressure, conductivity, turbidity, dissolved oxygen, chlorophyll: this list is not exhaustive and the number of sensors supported is increasing regularly. The RBR instruments have different channels available, one for each physical property measured (for example temperature) but also for physical properties derived from the measured ones (for example salinity). Channels can be referenced when sending commands via their indices (the first channel is at index 1) or via their label (for example temperature_00).

2.1.2 Acquiring samples

There are basically two fundamental ways to acquire samples. They can be acquired by fetching (polling) or according to a defined schedule.
Fetching measurement is performed via the command **fetch**.
Scheduled samples are configured with different commands (see **deployment** and **sampling**). They are always logged on board. They can be streamed directly out of the logger (see **streamusb** and **streamserial**). The underlying schedule can be as simple as a fixed sampling rate (see **continuous** mode in **sampling**) or more complex schemes (for example **ddsampling** or **regimes**). They can also obey some external control (**thresholding**, **twistactivation**).

One main feature of the RBR instruments is that fetching and scheduled samples can occur at the same time, they are not exclusive. The following quickstarts give examples of different sequences of commands and how to use the different methods to acquire samples.

2.2 Enabling continuous sampling

Start sampling for 24 hours at 1 sample per second starting from the current time of December 1, 2017 12:00 am.

```
>> clock datetime = 20171201000000
<< clock datetime = 20171201000000
>> deployment starttime = 20171201000000, endtime = 20171202000000
<< deployment starttime = 20171201000000, endtime = 20171202000000
>> sampling mode = continuous, period = 1000
<< sampling mode = continuous, period = 1000
>> verify
<< E0402 memory not empty, erase first
>> enable erasememory = true
<< enable status = logging, warning = none
>> deployment status
```

```
<< deployment status = logging
```

2.3 Enabling wave sampling

Collect 1024 samples at a rate of 4Hz every 30 minutes starting from December 2, 2017 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 3, 2017 12:00am.

```
>> clock datetime = 20171201000000
<< clock datetime = 20171201000000
>> deployment starttime = 20171202000000, endtime = 20171203000000
<< deployment starttime = 20171202000000, endtime = 20171203000000
>> sampling mode = wave, period = 250, burstlength = 1024, burstinterval = 1800000
<< sampling mode = wave, period = 250, burstlength = 1024, burstinterval = 1800000
>> permit command = settings
<< permit command = settings
>> settings altitude = 0.26
<< settings altitude = 0.2600
>> verify
<< E0402 memory not empty, erase first
>> enable erasememory = true
<< enable status = pending, warning = none
>> deployment status
<< deployment status = pending
```

2.4 Serial streaming from serial port

2.4.1 Setting the correct baud rate

Here we set the baud rate to 115200 via the **serial** command.

```
>> 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.
```

2.4.2 Enabling the serial streaming

An instrument will start streaming measurements as soon as it is logging (see **enable** command) and serial streaming is enabled (see **streamserial** command). If the instrument memory is full, the instrument will continue streaming as long as the instrument should be logging (see **deployment** command). The **outputformat** command indicates which channels will be reported and sets the type of output format to be used.
Here is an example for an RBR*duo*^{3} T.D:

```
% Set output format type to caltext01 (4 digits after decimal point without units)
>> outputformat type = caltext01
<< outputformat type = caltext01
% What channels will be reported?
>> outputformat labelslist
<< outputformat labelslist = temperature_00|pressure_00
```

```
% Enabling logging
>> enable erasememory=true
<< enable status = logging, warning = none
% Instrument does not stream but is logging
% Enabling serial streaming
>> streamserial state = on
<< streamserial state = on
% Instrument starts sending measurements
<< 2000-01-01 00:10:14.000, 10.0110, 501.3213
<< 2000-01-01 00:10:14.500, 10.0241, 501.0201
<< 2000-01-01 00:10:15.000, 10.0248, 500.8246
...
% Disabling logging
>> disable
<< disable
% Instrument does not stream nor log anymore
```

Here is another example, with a RBR*concerto*^{3} C.T.D:

```
% What channels will be reported?
>> outputformat labelslist
<< outputformat labelslist = temperature_00|pressure_00|salinity_00
% Set output format type to caltext02 (4 digits after decimal point with units)
>> outputformat type = caltext02
<< outputformat type = caltext02
% Enabling serial streaming
>> streamserial state = on
<< streamserial state = on
% At this moment, instrument does not log nor stream
% Enabling logging
>> enable erasememory=true
<< enable status = logging, warning = none
% Instrument starts sending measurements
<< 2000-01-01 00:10:14.000, 10.0110 C, 501.3213 dBar, 35.2012 PSU
<< 2000-01-01 00:10:14.500, 10.0241 C, 501.0201 dBar, 35.2013 PSU
<< 2000-01-01 00:10:15.000, 10.0248 C, 500.8246 dBar, 35.2001 PSU
...
% Disabling logging
>> disable
<< disable
% Instrument does not stream nor log anymore
```

2.5 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** section.

```
>> meminfo used
<< meminfo used = 2608
>> readdata dataset = 1, size = 500, offset = 0
<< readdata dataset = 1, size = 500, offset = 0<cr><lf><bytes[0…499]-of-data><crc>
>> readdata dataset = 1, size = 500, offset = 500
<< readdata dataset = 1, size = 500, offset = 500<cr><lf><bytes[500…999]-of-data><crc>
```

```
>> readdata dataset = 1, size = 500, offset = 1000
<< readdata dataset = 1, size = 500, offset = 1000<cr><lf><bytes[1000…1499]-of-
data><crc>
>> readdata dataset = 1, size = 500, offset = 1500
<< readdata dataset = 1, size = 500, offset = 1500<cr><lf><bytes[1500…1999]-of-
data><crc>
>> readdata dataset = 1, size = 500, offset = 2000
<< readdata dataset = 1, size = 500, offset = 2000<cr><lf><bytes[2000…2499]-of-
data><crc>
>> readdata dataset = 1, size = 500, offset = 2500
<< readdata dataset = 1, size = 108, offset = 2500<cr><lf><bytes[2500…2607]-of-
data><crc>
```

2.6 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.

Starting a deployment using EasyParse

Before the logger is enabled, it must be instructed to use the EasyParse format using the **memformat** 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 cannot be changed while logging is in progress.

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

This can be verified after the logger has been enabled, again using the **memformat** 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 Wi-Fi 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 Wi-Fi module was enabled').

Downloading the dataset

Downloading the data block in dataset-1 uses the **readdata** 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:

```
>> readdata dataset = 1, size = 2000, offset = 0
<< readdata dataset = 1, size = 2000, offset = 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)
```

There are just two channel readings in the data, so each sample is composed of a timestamp (an unsigned, 8-byte/64-bit integer number expressing milliseconds since the Unix epoch: 1970-01-01T00:00:00Z) followed by one reading for each of temperature and pressure (4-byte IEEE 754 floating-point numbers).

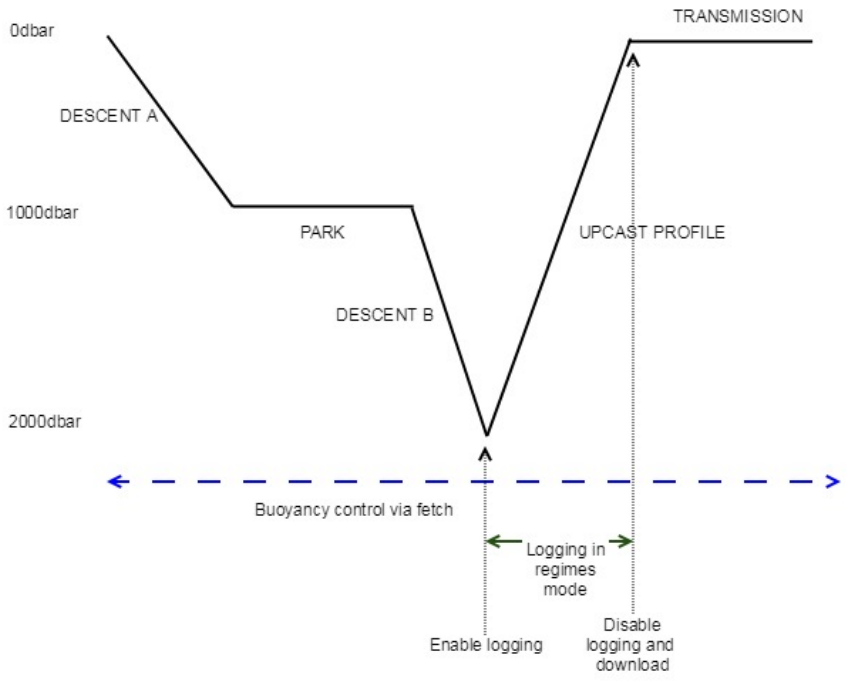
Converting the data at this point should be straightforward.

2.7 Integrating with a profiling float

2.7.1 Introduction

There are a number of dedicated features in the RBR L3 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:



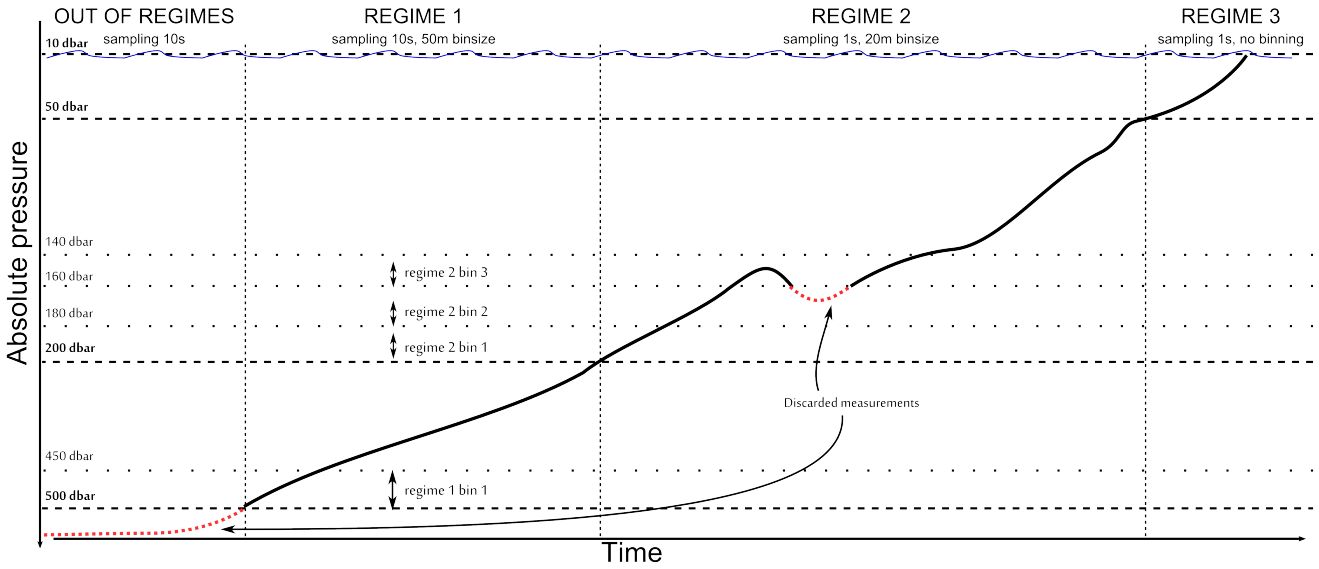
2.7.2 Buoyancy control

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** 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, channels = pressure_00** is used. This will override the idle timeouts and does not affect any ongoing deployment. It will also ensure, that the instrument is minimizing the power requirements by just powering and sampling the pressure sensor.

Always ensure the instrument is awake before sending the fetch by following the recommended wake-up procedure.

2.7.3 Setup for ascent, enable logging

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** and three **regime** 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 500dbar):

```
>> 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 200dbar):

```
>> 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 50dbar):

```
>> regime 3 boundary = 50, binsize = 0, samplingperiod = 1000
<< regime 3 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 can be shipped with a derived data channel populated, 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** 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 (calbin00).

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

Start the logger while ensuring the memory is cleared first.

```
>> enable erasememory = true
<< enable status = logging, warning = none
```

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

2.7.4 End of ascent, disable logging and download data

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:

```
>> disable
<< disable status = 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:

```
>> readdata dataset = 1, size = <chunk-size>, offset = 0
<< readdata dataset = 1, size = <chunk-size>, offset = 0<cr><lf> <bytes[offset…size]-of-
data><crc>
>> readdata dataset = 1, size = <chunk-size>
<< readdata dataset = 1, size = <chunk-size>, offset = <1 × chunk-size><cr><lf>
<bytes[offset…size]-of-data><crc>
>> readdata dataset = 1, size = <chunk-size>
<< readdata dataset = 1, size = <chunk-size>, offset = <2 × chunk-size><cr><lf>
<bytes[offset…size]-of-data><crc>
...
...
>> readdata dataset = 1, size = <chunk-size>
<< readdata dataset = 1, size = <final-chunk-size>, offset = <(n - 1) × chunk-
size><cr><lf> <bytes[offset…size]-of-data><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" section Briefly, each record consists of an unsigned, 8-byte/64-bit integer timestamp, and a 4-byte IEEE 754 floating-point number value for each channel.

2.7.5 More details on the calculation

Please note that samples taken in a bin where the outer limit has previously been exceeded are discarded, as shown in the example figure.

2.7.6 Available output channels

The list of channels populated onto an instrument and enabled by a float controller is subject to change depending on the float model (for example, some floats rely on hydrostatic pressure, others on absolute pressure) or revision (dynamic correction channels available only since fw 1.146).

Channel label	Description
conductivity_00	Conductivity (mS/cm)
temperature_00	Marine temperature (°C)
pressure_00	Absolute pressure (dbar) (at surface will read around 10 dbar)
seapressure_00	Hydrostatic pressure (dbar) (at surface will read around 0 dbar)
salinity_00	Salinity without dynamic correction applied (PSU)
conductivitycelltemperature_00	Internal temperature of the conductivity cell (°C)
temperaturedyncorr_00	Marine temperature corrected for C-T lag (°C), available since fw 1.146, see deri_dyncorrT and deri_dyncorrS dynamic correction channels
salinitydyncorr_00	Salinity with dynamic correction applied (PSU), available since fw 1.146, see deri_dyncorrT and deri_dyncorrS dynamic correction channels
cnt_00	Counts, number of sample used to calculate a bin average

2.7.7 Post-processing onboard

Starting from firmware 1.135, it is possible to bin the data after the ascent and to generate more statistics (like the standard deviation of samples) via the **postprocessing** command. The regimes mode is still relevant in conjunction with post-processing to retain a different sampling rate, but with the parameter binsize set to 0 to record all samples as the **postprocessing** command relies on data stored in dataset-1 to generate post-processed data.

```
% ---- set sampling parameters --------------------------

>> regimes direction = ascending, count = 3, reference = absolute
<< regimes direction = ascending, count = 3, reference = absolute
>> regime 1 boundary = 500, binsize = 0, samplingperiod = 10000
<< regime 1 boundary = 500, binsize = 0, samplingperiod = 10000
>> regime 2 boundary = 200, binsize = 0, samplingperiod = 1000
<< regime 2 boundary = 200, binsize = 0, samplingperiod = 1000
>> regime 3 boundary = 50, binsize = 0, samplingperiod = 1000
<< regime 3 boundary = 50, binsize = 0, samplingperiod = 1000
>> sampling mode = regimes
<< sampling mode = regimes
```

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

% ---- set postprocessing parameters --------------------------

>> postprocessing command=reset
<< postprocessing status = disabled

>> postprocessing mode = regimes
<< postprocessing mode = regimes
>> postprocessing_regimes direction = ascending, count = 3, reference = absolute
<< postprocessing_regimes direction = ascending, count = 3, reference = absolute
>> postprocessing_regime 1 boundary = 500, binsize = 50
<< postprocessing_regime 1 boundary = 500, binsize = 50.0
>> postprocessing_regime 2 boundary = 200, binsize = 20
<< postprocessing_regime 2 boundary = 200, binsize = 20.0
>> postprocessing_regime 3 boundary = 50, binsize = 0
<< postprocessing_regime 3 boundary = 50, binsize = 0.0

>> postprocessing channels =
mean(temperature_00)|mean(salinity_00)|mean(conductivitycelltemperature_00)|mean(tempera
turedyncorr_00)|mean(salinitydyncorr_00)
<< postprocessing channels =
mean(temperature_00)|mean(salinity_00)|mean(conductivitycelltemperature_00)|mean(tempera
turedyncorr_00)|mean(salinitydyncorr_00)
>> postprocessing command = start
<< postprocessing status = processing
>> enable erasememory = true
<< enable status = logging, warning = none
>> postprocessing status
<< postprocessing status = disabled
>> postprocessing command = enable
<< postprocessing status = processing
```

At the end of the ascent, stop logging, and wait for the post-processing to finish:

```
>> disable
<< disable status = stopped
>> postprocessing status
<< postprocessing status = processing
...
>> postprocessing status
<< postprocessing status = finished
```

And download the post-processed data (dataset-4):

```
>> meminfo dataset = 4, used
<< meminfo dataset = 4, used = <bytes-used-in-dataset-4>
>> readdata dataset = 4, size = <chunk-size>, offset = 0
<< readdata dataset = 4, size = <chunk-size>, offset = 0<cr><lf><bytes[offset…size]-of-
data><crc>
>> readdata dataset = 4, size = <chunk-size>
<< readdata dataset = 4, size = <chunk-size>, offset = <1 × chunk-
size><cr><lf><bytes[offset…size]-of-data><crc>

....
```

2.7.8 Providing platform details to end-users

Some end-users want to keep the sensor details history (for example, pressure sensor model). RBR Ltd. maintains a database of all instruments produced and their associated sensor. Providing the serial number (id command) should be sufficient in practice. But some end users might want to have all possible information readily available from the log files sent by the float to the shore.
The **getall** command outputs the result of all the other possible commands. If the output of that command is too large to be handled by the host, RBR recommends including the results from the **calibration**, **sensor**, **id**, and **info** commands in the log files.

2.7.9 Sensor drift monitoring at surface

It is possible to partly monitor the drift of different sensors when the float is at the surface and sensors are exposed to air. Pressure measurements at the surface give a direct offset correction. Optical dissolved oxygen measurements, air measurements, will also provide a reference for drift correction (see published literature). Measuring conductivity at the surface not only gives the opportunity to confirm that the float is effectively at the surface but also allows monitoring of any electronic drift in the conductivity as the sensor should read around 0. Conductivity in air measurements needs to be taken with precaution. It is advised to acquire several measurements in air and not just one as the conductivity cell might be washed by waves. If only one conductivity value is to be transmitted to shore, always use the lowest value. It is preferable to report directly the conductivity and not the salinity as the PSS78 calculation will saturate at 0 (see Example 7: pss78 - derivation of Practical Salinity (1978)).

2.7.10 Energy tracking

The instrument allows tracking of energy consumed via the command **powerexternal** .

2.8 High resolution BPR and frequency counters for cabled ocean observatories

2.8.1 Introduction

Loggers with frequency counter channels can achieve a very high measurement resolution (10 ppb). In order to maximize the resolution some additional considerations are required on the part of the user. This chapter presents the main points to follow to ensure the best performance for loggers using frequencies counters (and/or BPR channels). When sampling at a rate of 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 the integration time.

2.8.2 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 the **outputformat** command). The other calibrated output formats output 6 significant digits, which is insufficient to represent the full potential sensor resolution.

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 set up the deployment in continuous mode at 1Hz, and define the starttime and the endtime (here between the 2017-12-01 and the 2017-12-02):

```
>> sampling mode = continuous, period = 1000
<< sampling mode = continuous, period = 1000
>> deployment starttime = 20171201000000, endtime = 20171202000000
<< deployment starttime = 20171201000000, endtime = 20171202000000
```

High-resolution data must not be used in conjunction with the EasyParse memory format. The EasyParse memory format stores readings as single-precision (32-bits) floating-point numbers, which limits the achievable resolution.

We can now enable logging:

```
>> enable erasememory = true
<< enable status = logging, warning = none
>> deployment status
<< deployment status = logging
```

The instrument now continuously outputs the measurements:

```
<< 2017-12-01 18:37:48.000, 30.39588279090822e+006, 5.825177871156484e+006,
17.1028520e+000, 22.4525380e+00
```

2.8.3 BPR channels

High-resolution RBR BPR instruments are configured to provide 2 frequency counter channels and 2 BPR channels per BPR sensor: pressure signal period, temperature signal period, calculated pressure, calculated temperature. The instrument stores only the frequency measurements (BPR channels being derived).

The **channel** command gives back:

```
>> channel 1
<< channel 1 type = peri00, module = 96, status = on, settlingtime = 900, readtime =
858, equation = lin, userunits = ps, label = none
>> channel 2
<< channel 2 type = peri01, module = 97, status = on, settlingtime = 900, readtime =
858, equation = lin, userunits = ps, label = none
>> channel 3
<< channel 3 type = bpr_08, module = 243, status = on, settlingtime = 0, readtime = 0,
equation = lin, userunits = dbar, label = none
>> channel 4
<< channel 4 type = bpr_09, module = 244, status = on, settlingtime = 0, readtime = 0,
equation = lin, userunits = C, label = none
```

And the **calibration** command returns:

```
>> calibration 1
<< calibration 1 label = none, type = peri00, datetime = 20170401000000, c0 =
20.000000e+006, c1 = 10.000000e+006
```

```
>> calibration 2
<< calibration 2 label = none, type = peri01, datetime = 20170401000000, c0 =
5.0000000e+006, c1 = 2.5000000e+006
>> calibration 3
<< calibration 3 label = none, type = bpr_08, datetime = 20171123120721, 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 =
58.803408e+000, x9 = 180.91160e+000, x10 = 0.0000000e+000, n0 = 1, n1 = 2
>> calibration 4
<< calibration 4 label = none, type = bpr_09, datetime = 20171123120722, 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 15: deri_bprpres and deri_bprtemp, BPR channels in order to change the calibration.

2.8.4 Operating an RBR*quartz*^{3} BPR|zero instrument

RBR*quartz*^{3} BPR|zero instruments are able to compensate the drift of the pressure sensor by regularly measuring the internal pressure of the instrument body with the same pressure sensor and comparing it to a reference barometer placed inside the instrument.
In order to do so, the instrument operates a valve to switch the pressure sensor between being exposed to the sea pressure (Marine) and being exposed to the instrument housing pressure (Atmospheric).
In order to set up the deployment so that the RBR*quartz*^{3} BPR|zero instrument regularly operates the valve, the following steps are required prior to enabling the deployment (see details of command **valve**) .

If one wants to perform a measurement of the internal pressure every month for 5 minutes:

```
>> valve scheduled = true
<< valve scheduled = true
>> valve interval = 2678400000
<< valve interval = 2678400000
>> valve duration = 300000
<< valve duration = 300000
```

Then enable the deployment (see details above in the case of a BPR instrument):

```
>> enable erasememory = true
<< enable status = logging, warning = none
>> deployment status
<< deployment status = logging
```

If one wants to trigger a measurement of the internal pressure (Atmospheric) manually:

```
>> valve command = setpositionA
<< valve status = positionA
```

And setting the instrument back to the sea pressure (Marine) measuring position:

```
>> valve command = setpositionM
<< valve status = positionM
```

If the last step is omitted, the valve will return to the Marine position (**positionM**) at the end of the next scheduled episode.

2.9 Migrate from L2 to L3 platform

2.9.1 Introduction

For customers who have used the L2 platform, migrating to the L3 platform will require some substantial changes in any interfacing software used. Some commands have been changed, and some new commands have been introduced. This quick-start section is intended to help guide users through these changes to ensure a smooth transition.

A few L2 commands have been deprecated, which means that they are no longer documented, but may still exist on the L3 platform to help users through the transition period. However, there is no guarantee that this will always be the case, so we **strongly recommend that their use be discontinued as soon as possible**.

2.9.2 Identifying the L3 platform

The L3 platform can be easily identified via the **fwtype** parameter of the id command.

In the case of the L3 platform, one would obtain:

```
>> id fwtype
<< id fwtype = 104
```

Whereas, in the case of the L2 platform, one would obtain:

```
>> id fwtype
<< id fwtype = 103
```

2.9.3 Deprecated commands

The following commands have been deprecated:

- **stop**: replaced by the **disable** command
  - With L2 platform:

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

  - With L3 platform:

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

- **now**: replaced by the **clock** command: see also the notes below about the **clock** command for other options
  - With L2 platform:

    ```
    >> now
    << now = 20170901120000
    ```

  - With L3 platform:

    ```
    >> clock datetime
    << clock datetime = 20170901120000
    ```

- **starttime**: replaced by the **deployment** command
- **endtime**: replaced by the **deployment** command
- **status**: replaced by the **deployment** command
    - With L2 platform:

```
>> starttime
<< starttime = 20000101000000
>> endtime
<< endtime = 20991231235959
>> status
<< status = stopped
```

    - With L3 platform:

```
>> deployment
<< deployment starttime = 20000101000000, endtime = 20991231235959, status =
stopped
```

- **powerstatus**: replaced by the **power** command. See also **powerinternal** and **powerexternal** commands
    - With L2 platform:

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

    - With L3 platform:

```
>> power
<< power source = usb, int = 12.40, ext =  0.00
```

- **read**: replaced by the **readdata** command
    - With L2 platform:

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

    - With L3 platform:

```
>> readdata dataset = 1, size = 1000, offset = 2000
<< readdata dataset = 1, size = 12, offset = 2000<cr><lf><bytes[2000..2011]-of-
data><crc>
```

2.9.4 Removed commands

- **errorlog**
- **altitude**: replaced by the **settings** command.
    - With L2 platform:

```
>> altitude
<< altitude = 150.0
>> altitude = 0.26
<< altitude = 0.26
```

- With L3 platform:

```
>> settings altitude
<< settings altitude = 150.0000
>> permit command = settings
<< permit command = settings
>> settings altitude = 0.26
<< settings altitude = 0.2600
```

2.9.5 Improved commands and new parameters

- **fetch**: the fetch command allows now a specific subset of **channels** to be sampled.
  - With L3 platform:

```
>> fetch channels = 3|2
<< 2017-10-21 11:50:49.000, 12.7052 dbar, 18.1745 C
```

- **powerinternal** and **powerexternal**: track power usage.
  - With L3 platform:

```
>> powerinternal
<< powerinternal batterytype = nimh, capacity = 138.0e+003, used = 100.1e+003
>> powerexternal
<< powerexternal batterytype = fermata_lisocl2, capacity = 22.0e+006, used =
100.1e+003
```

- **wifi**: new parameters have been added.
  - With L2 platform:

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

  - With L3 platform:

```
>> wifi
<< wifi enabled = false, state = n/a, timeout = 120, commandtimeout = 60,
baudrate = 921600
```

- **sampling**: **availablefastperiods** parameter added.
  - With L3 platform:

```
>> sampling availablefastperiods
<< sampling availablefastperiods = 500|250|125|63
```

- **outputformat**: **support** parameter renamed to **availabletypes**. Now uses pipe separation between list items.
  - With L2 platform:

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

  - With L3 platform:

```
>> outputformat availabletypes
<< outputformat availabletypes = caltext01|caltext02|caltext03|caltext04
```

  - With L2 platform:

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

  - With L3 platform:

```
>> outputformat channelslist
<< outputformat channelslist =
temperature(C)|pressure(dbar)|pressure(dbar)|depth(m)
```

- **deployment** replaces the **starttime**, **endtime**, and **status** commands
- **enable**: status is now reported as a key-value pair, and warnings reported via a parameter. Command does not fail if logging is already active.
  - With L2 platform:

```
>> enable erasememory = true
<< E0401 estimated memory usage exceeds capacity, enable = logging
```

  - With L3 platform:

```
>> enable erasememory = true
<< enable status = logging, warning = none
>> enable erasememory = true
<< enable status = logging, warning = W0408
```

- **disable**: status is now reported as a key-value pair. Command does not fail if logging not active.
  - With L2 platform:

```
>> disable
<< disable = stopped
```

  - With L3 platform:

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

- **verify**: status is now reported as a key-value pair and warning parameter added as well.
  - With L2 platform:

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

  - With L3 platform:

```
>> verify
<< verify status = logging, warning = none
```

- **clock**: reports and sets the date and time as the **datetime** parameter and incorporates the UTC offset setting as the **offsetfromutc** parameter.
  - With L2 platform:

```
>> now
<< now = 20170901120000
>> now = 20170901155450
<< now = 20170901155450
```

  - With L3 platform:

```
>> clock
<< clock datetime = 20170901120000, offsetfromutc = +1.20
>> clock datetime = 20170901155450
<< clock datetime = 20170901155450
```

- **link**: type is now reported as a key-value pair.
  - With L2 platform:

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

  - With L3 platform:

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

- **permit**: command is now received as a key-value pair.
  - With L2 platform:

```
>> permit memclear
<< permit = memclear
```

  - With L3 platform:

```
>> permit command = memclear
<< permit command = memclear
```

- **memformat**: **support** parameter renamed to **availabletypes**, uses pipe separation between list items.
  - With L2 platform:

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

  - With L3 platform:

```
>> memformat availabletypes
<< memformat availabletypes = rawbin00|calbin00
```

- **channels**: **latency** parameter renamed to **settlingtime.**
  - With L2 platform:

```
>> channels latency
<< channels latency = 300
```

  - With L3 platform:

```
>> channels settlingtime
<< channels settlingtime = 300
```

- **channel**: **all** keyword replaced by **allindices** and **alllabels**. Added channel labels.
  - With L2 platform:

```
>> channel all type
<< channel 1 type = temp01 | channel 2 type = pres24
```

  - With L3 platform:

```
>> channel all type
<< E0108 invalid argument to command: 'all'
>> channel allindices type
<< channel 1 type = temp01 || channel 2 type = pres24
>> channel alllabels type
<< channel temperature_00 type = temp09 || channel pressure_00 type = pres24
```

- **Calibration**: **all** keyword replaced by **allindices** and **alllabels**. Added channel labels.
  - With L2 platform:

```
>> calibration all datetime
<< calibration 1 datetime = 20171220161738 | calibration 2 datetime =
20171220161738
```

  - With L3 platform:

```
>> calibration all datetime
<< E0108 invalid argument to command: 'all'
>> calibration allindices datetime
<< calibration 1 datetime = 20171220161738 | calibration 2 datetime =
20171220161738
>> calibration alllabels datetime
<< calibration temperature_00 datetime = 20171220161738 | calibration pressure_00
```

```
datetime = 20171220161738
```

- **sensor**: support allindices and alllabels keywords
- **getall**: command added
- **info**: command added

2.9.6 Removed parameters

- **sampling schedule**
- **calibration type**

2.9.7 Syntactical changes

- Reference to channel via labels

```
>> fetch channels = pressure01|temperature01
<< 2017-10-21 11:50:49.000, 12.7052 dbar, 18.1745 C
```

- || instead of | for **all, allindices, alllabels** keywords.
  - With L2 platform:

```
>> channel all type
<< channel 1 type = temp09 | channel 2 type = cond06
```

  - With L3 platform:

```
>> channel alllabels type
<< channel temperature_00 type = temp09 || channel conductivity_00 type = cond06
>> channel allindices type
<< channel 1 type = temp09 || channel 2 type = cond06
```

- Besides the command relying on **alllabels** and **allindices**, all commands now support the **all** keyword

2.10 Tips for system integrators

2.10.1 Default deployment start and end time

The command deployment sets the start and end time of a deployment. This concept of start/end time is beneficial mainly to standard products users. In the context of an integration with a host controller, it is generally useless, scheduling of logging or streaming being controlled directly by the host controller. Furthermore, on some system the clock might just be discarded and timestamping applied by host controller (streaming instruments). In order to not have to set correctly the start and end time relative to the onboard clock, one simple solution is to set them as follows:

```
>> deployment starttime = 20000101000000 endtime = 20991231235959
```

The onboard RTC clock minimum date time is 2000/01/01 00:00:00 and the maximum date time is 2099/31/12 23:59:59.

2.10.2 Sampling rates

The **sampling** command allows the user to set faster than 1Hz sampling rate by specifying a number of milliseconds

below 1000. It also reports a list of milliseconds values (see **availablefastperiods** parameter). As there is in many cases no direct conversion between an integer number of milliseconds and a frequency in Hertz, the following explains how to convert from one to another.

Converting from Hz to milliseconds

If Fs is the sampling rate in Hz, then the number of milliseconds to be used as a parameter for the **sampling** command, is calculated as (using integer division):

$$Ts = \frac{1000 + \frac{Fs}{2}}{Fs}$$

Converting from milliseconds to Hz

If Ts is the number of milliseconds reported by the logger as the period (see **sampling** command), then the corresponding sampling rate Fs in Hz, is calculated as (using integer division):

$$Fs = \frac{1000 + \frac{Ts}{2}}{Ts}$$

Examples

If the logger needs to be deployed at 13Hz, and this sampling rate is available, then the period to be used would be 77 ms. If the logger is deployed with a period of 53 ms, the effective sampling rate is 19Hz.

2.10.3 Future proofing development

The chapter Command processing and timeouts gives a good overview on best practices on how to parse and handle logger commands.

For OEM customers using more than one configuration or planning to, it is good practice to not rely on a fixed channel order (between a T.D configuration and a C.T.D configuration, the temperature channel is not necessarily at the same index). In order to identify available channels on a platform, the best way is to use labels (see **channel**). It is also not recommended to identify channels via their channel type as they might change for the same configuration over time. Those are generated at RBR factory and correspond to the physical properties measured.

In order to process all the commands, RBR suggests using a command buffer of 1Kbyte, as it will cover most commands with the exception of **getall**, **help** and, when all channels are requested, **sensor**, **channel**, **calibration**.

2.10.4 Power management and power cycling behaviour

The RBR logger can accommodate two power sources: internal batteries (see **powerinternal**) or external power source (see **powerexternal**).

The RBR logger handles power management automatically, switching automatically to one or the other power source depending on power availability. The RBR L3 also handles automatically which power domains should be enabled or not depending on its current state (sampling, asleep, in communications). In systems which power cycle the logger, a full reset will be obtained by powering off the unit at least **5 minutes** due to internal capacitors.

There is no dedicated battery to maintain the onboard clock. As a consequence, the onboard clock might be lost every time there is no power source available. In such case the date and time will reset at 2000-01-01 00:00:00. If the logger endures a reset while logging, and the clock is lost, the logger will continue logging but timestamps might restart at 2000-01-01 00:00:00 even if the starttime (see deployment) is set to another date/time.

Upon power cycling the instrument may take up to 4 seconds to initialize. During that initialization period, characters transmitted to the instrument might not be received, and commands not processed correctly.

If an instrument is power cycled while it was logging or streaming, it will continue streaming and logging (even if the RTC clock is reset). In the case of an instrument streaming, measurements will start to be streamed after the settling time and readtime periods expire (see **channels**).

In order to enforce a full hardware reset, the following sequence should be followed:

1. power off the unit
2. wait 5 minutes
3. power on the unit
4. wait 4 seconds before sending any commands (initialization time)

When the instrument is not powered, all UART and RS232 lines must be in high impedance mode, with no pull-up resistors connected

2.10.5 Memory format

Except in the case of operating a BPR instrument in seismic application, the EasyParse "calbin00" format should be used.

2.10.6 Error handling

Instrument not responding

If the instrument stops responding to any commands, first apply a full hardware reset (as described above).
Send the command **id** followed by the command **disable**, if the instrument is still not responding repeat the same procedure with a baudrate of 115200 bps (default baudrate if configured baudrate lost).

Instrument reporting a hardware failure

If the instrument is reporting a hardware failure error code as described in Error messages, one course of action is to apply a full hardware reset (as described in Power management and power cycling behaviour).

Instrument reporting error codes in the measurements

Most of the error codes reported (see EasyParse "calbin00" format) reflect a hardware failure of some sort except for Error-14 which could just reflect a value outside of an equation.
Sometimes, this error can be transient and resolves by itself.
However, if the instrument is always reporting the same error for some period of time, one possible course of action is

to **disable** logging/scheduling, do not issue any **fetch** for 10 seconds, and **enable** again the unit.
If this does not resolve the issue, a full hardware reset (as described above), is advised. If one full hardware reset does not resolve the issue, it is unlikely that performing other full hardware reset will do.

In any case, only channels reported as Error should be discarded. A classical example is an instrument carrying cabled sensors. If a cable is damaged, the measurements associated with the sensor are likely to be reported as errors. Applying previous methods won't help and measurements from other channels are still valid and useful.

2.10.7 Electronic Static Discharge

Various electrical and electronic components are vulnerable to ESD. RBR PCBAs should be handled in a static controlled environment.

3 Commands

3.1 Time and schedule

3.1.1 pause

Usage

```
>> pause
```

Security

Open

Description

This command is available in firmware versions 1.116 or later. It pauses an enabled deployment.

If successful, the command reports its status:

- **status**, will be "paused".

Error conditions that would prevent the successful execution of the command are:

1. An older version of firmware that does not support the command is on the instrument.
2. The instrument has not been enabled for deployment.
3. A gating condition (twist, thresholding) is in effect for the deployment.
4. The instrument has been enabled in the 'regimes' sampling mode

When a pause is issued, the following will happen:

1. Streaming is immediately suspended if it was enabled for the deployment.
2. The current acquisition, if any, is terminated. For averages/tides, this means that the current sample will be thrown away and not recorded in memory or streamed. For bursts/waves the burst will terminate before the expected number of samples.
3. An event of type EVENT_PAUSE (0x2B) is stored in the data stream.
4. No further acquisitions will be scheduled until a **resume** command is received.

Examples

```
>> pause
<< pause status = paused
```

Deployment is now paused and no more samples will be taken once the current, if any, acquisition finishes.

```
>> pause
<< E0406 not logging
```

Cannot pause deployment before deployment is enabled.

```
>> pause
<< E0415 more than one gating condition is enabled
```

Assuming that either the wet switch or twist activation has been enabled the pause and resume feature cannot be used.

Errors

E0102 invalid command
The logger does not support the sampling pause and resume feature.

E0109 feature not available
The feature has not been allowed.

E0406 not logging
The logger deployment is not enabled and so cannot be paused/resumed.

E0415 more than one gating condition is enabled
If one gating mechanism has already been enabled (wet switch, twist) the pause and resume feature cannot be used.

E0417 no gating allowed with regimes mode
The logger deployment is enabled with the regimes sampling mode so the pause/resume feature cannot be used.

3.1.2 clock

Usage

```
>> clock [ datetime | offsetfromutc ]
```

Security

Unsafe.

Description

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

- **datetime** [= *<YYYYMMDDhhmmss>]*, reports or sets the current date and time.
- **offsetfromutc** [= *<+/-hh.hh>*] is intended to record the local timezone 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 **datetime** parameter; it is intended simply as a record of the local time zone. However, setting the date and time using the **datetime** parameter 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**.

Examples

```
>> clock
<< clock datetime = 20170401120000, offsetfromutc = +1.00
```

```
>> clock datetime = 20170401120130
<< clock datetime = 20170401120130
>> clock
<< clock datetime = 20170401120130, offsetfromutc = unknown
```

Errors

Error E0105 command prohibited while logging
Date/time may not be modified while logging is in progress; reading is permitted.

Error E0108 invalid argument to command: '<invalid-argument>'
The supplied argument was not a valid date/time.

3.1.3 deployment

Usage

```
>> deployment [ starttime | endtime | status ]
```

Security

Unsafe.

Description

Allows various parameters to be reported or set for the current deployment.

- **starttime** [=<*YYYYMMDDhhmmss*>], retrieve or set the start date and time of the next deployment.
- **endtime** [=<*YYYYMMDDhhmmss*>], retrieve or set the end date and time of the next deployment.
- **status** is a read-only parameter which returns the current state of the finite state machine for the instrument's logging function. Possible values are given below:
  - **disabled** : logging is not enabled.
  - **pending** : logging is enabled but before the start time.
  - **logging** : logging is in progress.
  - **paused**: logging paused, waiting for a resume command.
  - **gated** : logging paused, waiting for a gating condition to be satisfied.
  - **finished** : the programmed end time has been passed.
  - **stopped** : a **disable** 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 recovery procedures.

1. Send a **permit memclear** and **memclear** sequence, 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, 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** 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

```
>> deployment
<< deployment starttime = 20171214000000, endtime = 20171214000000, status = disabled
```

```
>> deployment starttime = 20171217120000
<< deployment starttime = 20171217120000
```

```
>> deployment status
<< deployment status = disabled
>> enable
<< enable status = logging
>> deployment status
<< deployment status = logging
```

Errors

Error E0105 command prohibited while logging
starttime and endtime cannot be modified while logging is in progress; reading is permitted.

Error E0108 invalid argument to command: '<invalid-argument>'
The supplied argument was not a valid date/time.

3.1.4 resume

Usage

`>> resume`

Security

Open

Description

This command is available in firmware versions 1.116 or later. It resumes an enabled deployment which was previously paused using the **pause** command.

If successful, the command reports its status:

- **status**, will be one of the values: "pending" or "logging".

Error conditions that would prevent the successful execution of the command are:

- An older version of firmware that does not support the command is on the instrument.
- The instrument has not been enabled for deployment.

- A gating condition (twist, thresholding) is in effect for the deployment.
- The instrument has been enabled in the 'regimes' sampling mode.

When a resume is issued, the following will happen:

1. Streaming is immediately activated if it was enabled for the deployment.
2. The next acquisition is scheduled for the appropriate time; in the case of average/burst/wave/tide, the next time will be aligned to the interval time.
3. An event of type EVENT_RESUME (0x2A) is stored in the data stream.
4. Acquisitions will continue to be scheduled at normal intervals.

Examples

```
>> resume
<< resume status = pending
```

Deployment has resumed running as scheduled.

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

Deployment has resumed running as scheduled.

Errors

E0102 invalid command
The logger does not support the sampling pause and resume feature.

E0109 feature not available
The feature has not been allowed.

E0406 not logging
The logger deployment is not enabled and so cannot be paused/resumed.

E0415 more than one gating condition is enabled
If one gating mechanism has already been enabled (wet switch, twist) the pause and resume feature cannot be used.

E0417 no gating allowed with regimes mode
The logger deployment is enabled with the regimes sampling mode so the pause/resume feature cannot be used.

3.1.5 pauseresume

Usage

`>> pauseresume`

Security

Open

Description

This command is available in firmware versions 1.116 or later. It allows the host to determine if the pause/resume feature is available on the instrument. It allows an elevated host to allow and deny the feature for the instrument.

If successful, the command reports its state:

- **state**, will be one of the values: "n/a", "paused", or "running".

Error conditions that would prevent the successful execution of the command are:

- An older version of firmware that does not support the command is on the instrument.
- The permission for the feature has not been allowed.

Examples

```
>> pauseresume
<< E0109 feature not available
```

The feature is not allowed on this instrument.

```
>> pauseresume
<< E0102 invalid command 'pauseresume'
```

The instrument firmware is older and does not recognize this feature.

```
>> pauseresume
<< pauseresume state = n/a
```

Either the deployment has not been enabled or the sampling mode is 'regimes'.

```
>> pauseresume
<< pauseresume state = running
```

The deployment has been enabled and is not paused.

```
>> pauseresume
<< pauseresume state = paused
```

The deployment has been enabled and is paused.

Errors

E0102 invalid command

The logger does not support the sampling pause and resume feature.

E0109 feature not available

The feature has not been allowed.

3.1.6 sampling

Usage

```
>> sampling [ mode | period | availablefastperiods | userperiodlimit |
burstinterval | burstlength | gate ]
```

Security

Unsafe.

Description

Allows various parameters to be reported or set for the next deployment. The <*parameter*>s currently supported are:

- **mode** [= <*mode*>] reports or sets the sampling mode for the current schedule:
  - **continuous** mode is supported by all loggers; measurements are taken and stored at the specified period between the start and end times.
  - **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**.
  - **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.
  - **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.
  - **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.
  - **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.
  - **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 current direction of the logger (descending or ascending). See command **ddsampling**.

> Use of directional dependent sampling requires the logger to have a suitable sensor for measuring pressure. Firmware versions 1.135 or later will check this condition when an attempt is made to set **ddsampling** mode. If no suitable sensor is available the following error message is returned:
>
> **Error E0114**
>
> For reasons relating to the logger's internal operation, not all types of pressure sensor are suitable for use; for example, high precision quartz sensors cannot be used to control directional dependent sampling.

- **period** [= <*period*>] reports or sets the time between measurements.
  This is straightforward 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 are further constrained to a supported subset of sample rates, which may be determined via the **availablefastperiods** parameter described below. See also Tips for system integrators.

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

- **availablefastperiods** reports a list of the fast measurement periods available for the logger for sampling rates faster than 1Hz. Each available period is reported to the nearest millisecond, and the values are separated by a vertical bar, or 'pipe' character, '|'. If there are no fast periods available, the word **none** is returned instead of a list of values. When a list is reported, the **period** for sampling rates faster than 1Hz can be set to any value in the list, but no others.

> The periods given in the list may be supported in some modes but not others, depending on the logger's configuration.

- **userperiodlimit** reports the minimum period which can be used in 'fast' sampling modes, in milliseconds; the **period** cannot be set to a value less than this.
- **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:
  - **none**: no gating conditions are enabled.
  - **thresholding**: a threshold requirement must be satisfied; see the **thresholding** command.
  - **twistactivation**: the end-cap must be in the "on" position; see the **twistactivation** command.
  - **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.

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

- **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**.
- **burstlength** [= <*burstlength*>] reports or sets the number of measurements taken in each burst.
  The permitted range is 2 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**).

Examples

```
>> sampling all
<< sampling mode = continuous, period = 125, burstlength = 60, burstinterval = 300000,
gate = none, userperiodlimit = 63, availablefastperiods = 500|250|125|63
```

The logger has been programmed for continuous 8Hz sampling. It supports 2Hz, 4Hz, 8Hz, and 16Hz operation. 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 mode = average, period = 125, burstlength = 60, burstinterval = 300000, gate
= none, userperiodlimit = 63
```

Averaging enabled without changing the other parameters; the logger is now programmed to take a burst of 60 measurements at 8Hz every five minutes, and store the average of the 60 measurements.

```
>> sampling mode = burst, burstinterval = 600000
<< sampling mode = burst, burstinterval = 600000
>> sampling
<< sampling mode = burst, period = 125, burstlength = 60, burstinterval = 600000, gate =
none, userperiodlimit = 63
```

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 8Hz every ten minutes, storing all measurements in memory.

```
>> sampling
<< sampling mode = continuous, period = 250, gate = none, userperiodlimit = 125
```

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.

```
>> sampling availablefastperiods
<< sampling availablefastperiods = 500|250|125|63

>> sampling availablefastperiods
<< sampling availablefastperiods = none
```

The first example shows support for sampling at 2Hz, 4Hz, 8Hz and 16Hz in at least one of the available modes; the second example shows that no sampling faster than 1Hz is supported.

Errors

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

Error E0108 invalid argument to command: '<invalid-argument>'
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 feature not available
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.

Error E0114 feature not supported by hardware
The logger does not have a pressure sensor suitable for controlling the directional dependent sampling mode.

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.

- Thresholding.
- Twist activation.

3.2.1 thresholding

Usage

```
>> thresholding [ enabled | state | channelindex | channellabel | condition | value | interval ]
```

Security

Unsafe.

Description

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

- **enabled** [= <*enabled*>]
  - **true**, enables the threshold gating feature.
  - **false**, disables the threshold gating feature.
- **state** is a read-only parameter which shows whether logging is currently **paused** or **running** due to the thresholding configuration, or **n/a** because thresholding is not enabled or because logging has not yet been enabled.
- **channelindex** [= <*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.

- **channellabel** [= <*label*>] the label 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.
- **condition** [= <condition>]
  - **above**, sampling will occur when the monitored parameter is above the specified threshold value.
  - **below**, sampling will occur when the monitored parameter is below the specified threshold value.
- **value** [= <*value*>] specifies the threshold value in calibrated units, given as a number compatible with floating point formats.
- **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 enabled = false, state = n/a, channelindex = 1, channellabel =
temperature_01, 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 enabled = true
<< thresholding enabled = true
```

Threshold gated control of sampling is enabled.

Errors

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

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

Error E0108 invalid argument to command: '<invalid-argument>'
The command was given with an argument which is unrecognized or has an invalid value; for example "thresholding enabled = yes", or "thresholding interval = 10".

Error E0601 no calibration for channel '<channel-index>'
The channel selected for threshold checking is not calibrated.

3.2.2 twistactivation

Usage

```
>> twistactivation [ enabled | state ]
```

Security

Unsafe.

Description

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

- **enabled** [**= true | false**]
  - **true**, enables the twist activation gating feature.
  - **false**, disables the twist activation gating feature.
- **state** is a read-only parameter which shows whether logging is currently **paused** or **running** due to the position of

the end cap. The **state** may also be reported as **n/a** because

- Twist activation is not enabled.
- The logger has not yet been enabled for sampling.
- The logger is enabled but still before the start time (**deployment status = pending**).

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, thresholding and twist activation gating features may not be enabled simultaneously.

Examples

```
>> twistactivation
<< twistactivation enabled = false, state = n/a
```

Twist-activation-gated logging is not enabled. If it were, the logger would sample data depending on the position of the end cap.

```
>> twistactivation enabled = true
<< twistactivation enabled = true, state = n/a
>> enable
<< enable status = gated, warning = none
>> twistactivation
<< twistactivation enabled = true, state = paused
```

Twist-activation-gated control of sampling is enabled, and logging is paused pending twist activation.

Errors

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

Error E0108 invalid argument to command: '<invalid-argument>'
The command was given with an argument which is unrecognized or has an invalid value; for example "twistactivation enabled = yes".

Error E0109 feature not available
The logger is not configured to support twist-activation-gated control of sampling.

3.3 Vehicle support

3.3.1 regimes

Usage

```
>> regimes [ direction | count | reference ]
```

Security

Unsafe.

Description

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

- **direction** [= <*direction*>]
  - **ascending** indicates that the instrument is intended to ascend in the water column.
  - **descending** indicates that the instrument is intended to descend in the water column.

Note: 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.

- **count** [= <*countvalue*>] indicates the number of regimes that are set. The minimum number is 1 and the maximum number of regimes is 3.
- **reference** [= <*reference*>]
  - **absolute** indicates that the absolute pressure is used as reference for the determination of the current regime and bin.
  - **seapressure** indicates that the sea 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 command prohibited while logging
Settings may not be modified while logging is in progress; reading them is permitted.

Error E0108 invalid argument to command: '<invalid-argument>'
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 feature not available
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

Unsafe.

Description

Returns information about the specified <*index*> regime. The first regime has an <*index*> of 1. <index> should be lower or equal to the number or regimes reported by the **regimes** command at the time of issuing the command. If **direction** (**regimes** 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** 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.

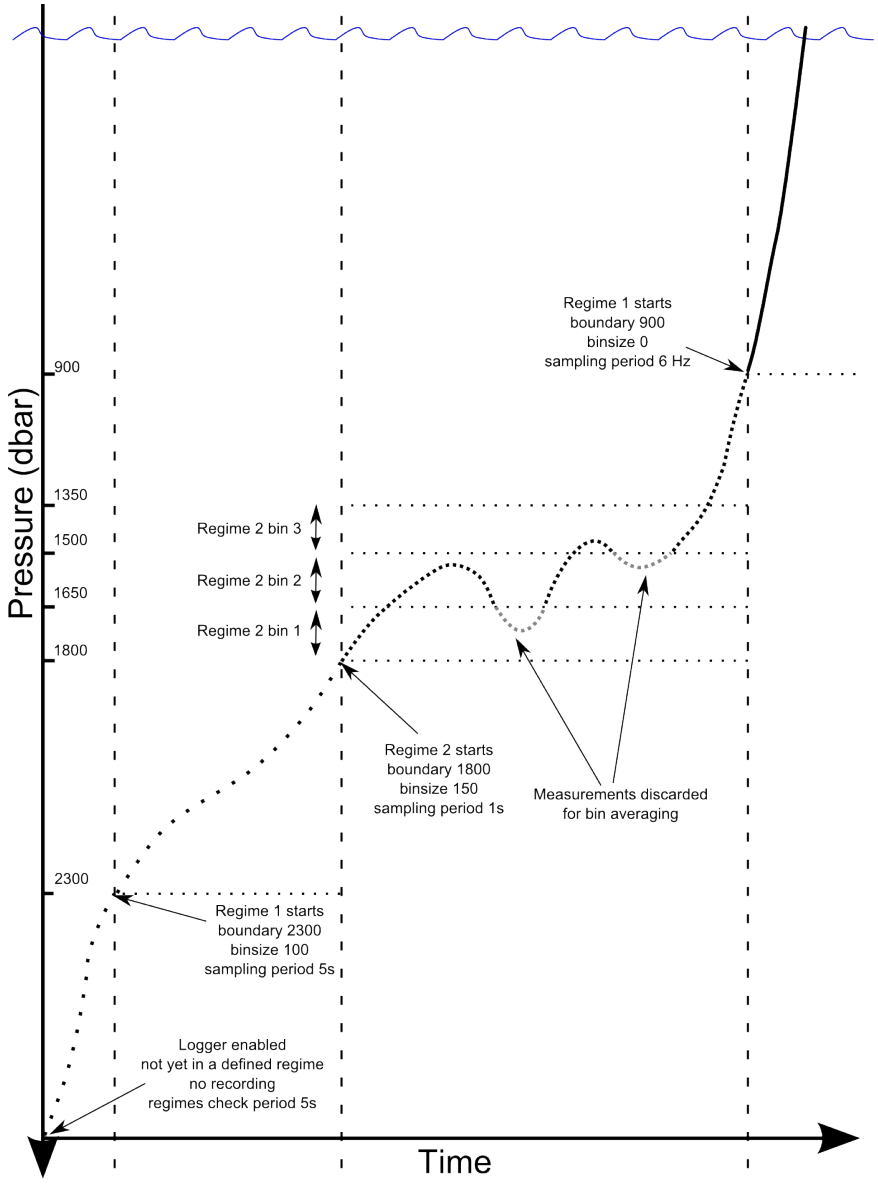
- **boundary** [= <*firstboundaryvalue*>] specifies the transition from one regime to the next, and is interpreted as the *first* boundary in a region. The units are in dbar. The minimum precision is 1 dbar, and the value should be between 0 dbar and 65535 dbar. **verify** and **enable** commands will check that the regimes boundaries are strictly increasing if **direction** is descending and strictly decreasing if **direction** is ascending, reporting an error **E0416** for invalid regime settings if those constraints are violated.
- **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.
- **samplingperiod** [= <*period*>] has the same meaning as the **period** value in the **sampling** 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 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.



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** 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** command is issued.

Errors

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

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

Error E0108 invalid argument to command: '<invalid-argument>'
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 feature not available
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

Unsafe.

Description

Sets and returns information about the directional dependent sampling mode. These settings apply only if the ddsampling mode is in active use (see **sampling**). This mode is intended for use when the instrument is integrated into a moving platform.

- **direction** [= <*direction*>]
  - **ascending** indicates that the instrument should sample at the fast rate during ascents.
  - **descending** indicates that the instrument should sample at the fast rate during descents.
- **fastperiod** [= <*period*>] has the same meaning as the period value in the **sampling** 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.
- **slowperiod** [= <*period*>] has the same meaning as the period value in the **sampling** 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.
- **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.

- **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). 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:

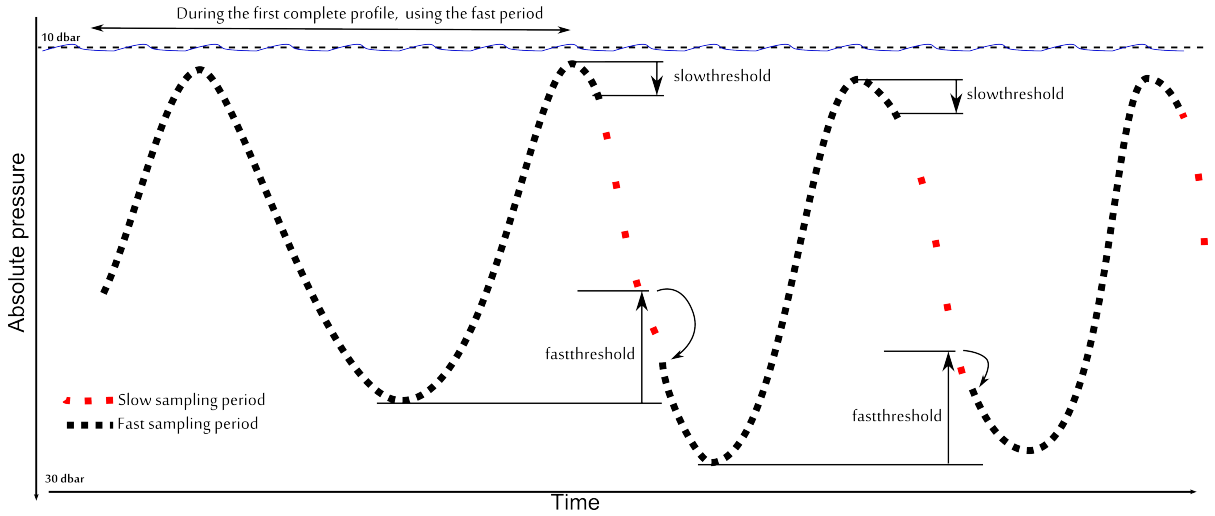
- **fastthreshold** should be greater than maxspeed x (**slowperiod** + modetransitiontime) + maxprofilevariation where:
  - modetransitiontime is the sum of the settling and read times as reported by the **channels** command, plus one second (blanking period while switching sampling rate).
  - maxspeed 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).

> Use of directional dependent sampling requires the logger to have a suitable sensor for measuring pressure. Firmware versions 1.135 or later will check this condition and respond to the **ddsampling** command with this error message if no suitable sensor is available:
>
> **Error E0114**
>
> For reasons relating to the logger's internal operation, not all types of pressure sensor are suitable for use; for example, high precision quartz sensors cannot be used to control directional dependent sampling.

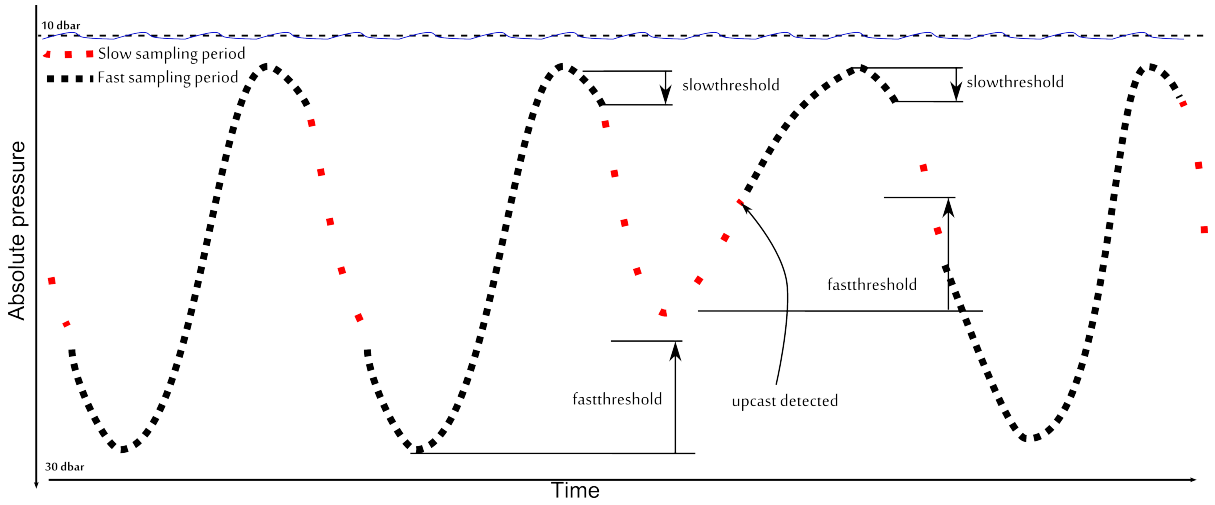
Examples

```
>> 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 command prohibited while logging

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

Error E0108 invalid argument to command: '<invalid-argument>'

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 feature not available

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

Error E0114 feature not supported by hardware

The logger does not have a pressure sensor suitable for controlling the directional dependent sampling mode.

3.4 Deployments

3.4.1 verify

Usage

```
>> verify [ erasememory = true ]
```

Security

Open.

Description

This command performs all the same deployment 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.

As with the **enable** command, it takes the option:

- **erasememory = true**

As with the **enable** command, it reports two parameters:

- **status**, this is the same **status** reported by the **deployment** command, as it would be reported by the **enable**.
- **warning**, warning code if any (see description below), **none** otherwise.

```
>> verify
<< verify status = pending, warning = none
```

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

```
>> verify
<< verify status = logging, warning = none
```

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

```
>> verify
<< E0402 memory not empty, erase first
>> verify erasememory=true
<< verify status = logging, warning = W0401
```

Memory is not cleared, logging would not be enabled. Except in the case of the **erasememory = true** option.

Warning messages:

W0401 estimated memory usage exceeds capacity

The schedule is valid, but the memory will fill up before the end time is reached.

W0408 logging already active

The logger was already active.

Errors

Error E0402 memory not empty, erase first
The data memory must be empty. Use either the **memclear** command or the **erasememory** parameter of the **enable** command to erase data memory.

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

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

Error E0410 no sampling channels active
All the logger's measured channels have been turned off (in instruments which support this feature); there is nothing to sample.

Error E0411 period not valid for selected mode
The measurement period is not consistent with other programmed parameters.

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

Error E0413 period too short for serial streaming
Serial streaming may not be available at high sample rates.

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

Error E0415 more than one gating condition is enabled
Gating conditions such as thresholding and twist activation cannot be combined; disable all but one gating condition.

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

Error E0417 no gating allowed with regimes mode
A gating condition cannot be used with the "regimes" sampling mode.

Error E0418 cast detection needs a pressure/depth channel
The cast detection feature cannot be used if the logger has no pressure or depth channel.

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

Error E0420 required channel is turned off; <channel-index>
The indicated channel has been turned off, but it must be enabled in order for a selected feature to work; turn the channel back on.

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

Error E0422 AUX1 not available in current serial mode
The AUX1 signal can be used only in RS-232 mode.

Error E0423 wrong ddsampling settings
The programmed settings for the "ddsampling" sampling mode are not consistent.

Error E0114 feature not supported by hardware
An attempt is being made to enable the logger using a feature it does not support.

3.4.2 enable

Usage

```
>> enable [ erasememory ]
```

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.

- **erasememory = true**, erase the memory

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 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.

If successful, the command reports two parameters:

- **status**, this is the same **status** reported by the **deployment** command.
- **warning**, warning code if any (see description below), **none** otherwise.

Examples

```
>> enable
<< enable status = pending, warning = none
```

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

```
>> enable
<< enable status = logging, warning = none
```

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

```
>> enable
<< enable status = pending, warning = W0401
```

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:

W0401 estimated memory usage exceeds capacity
The schedule is valid, but the memory may fill up before the end time is reached.

W0408 logging already active
The logger was already active.

Errors

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

Error E0108 invalid argument to command: '<invalid-argument>'
An unrecognized argument was given with the command.

Error E0301 memory erase not completed
Memory erase failed (requested with **erasememory** = **true** option).

Error E0402 memory not empty, erase first
The data memory must be empty

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

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

Error E0405 failed to enable for logging
An internal problem prevented the logger from enabling the sampling schedule. For further details and suggestions for recovery, refer to the **deployment** command.

Error E0410 no sampling channels active
All the logger's measured channels have been turned off (in instruments which support this feature); there is nothing to sample.

Error E0411 period not valid for selected mode
The measurement period is not consistent with other programmed parameters.

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

Error E0413 period too short for serial streaming
Serial streaming may not be available at high sample rates.

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

Error E0415 more than one gating condition is enabled
Gating conditions such as thresholding and twist activation cannot be used in combination; choose one.

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

Error E0417 no gating allowed with regimes mode
A gating condition cannot be used with the 'regimes' sampling mode.

Error E0418 cast detection needs a pressure/depth channel
The cast detection feature cannot be used if the logger has no pressure or depth channel.

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

Error E0420 required channel is turned off; <channel-index>
The indicated channel has been turned off, but it must be sampled for a selected feature (e.g., thresholding) to work; turn the channel back on.

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

Error E0422 AUX1 not available in current serial mode
The AUX1 signal can be used only in RS-232 mode.

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

Error E0114 feature not supported by hardware
An attempt is being made to enable the logger using a feature it does not support.

3.4.3 disable

Usage

```
>> disable
```

Security

Open.

Description

If the instrument is logging, this command will terminate the current deployment. If the instrument was not logging when the command is issued, its status does not change.
This command reports:

- **status**, the status of the logger is always reported when the command is complete (see **deployment** command).

If the **disable** 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.

Examples

```
>> deployment status
<< deployment status = logging
>> disable
```

```
<< disable status = stopped
```

The instrument was logging, and has now been stopped.

```
>> deployment status
<< deployment status = disabled
>> disable
<< disable status = disabled
```

The instrument had previously been stopped and its data memory erased; its status has not changed.

3.4.4 simulation

Usage

```
>> simulation [state | period ]
```

Security

Unsafe and protected.

Description

This command controls whether the simulation mode is used, and if so how it will operate. It is an Unsafe command, meaning that parameters cannot be modified while the instrument is enabled for logging. Modification of parameters is also Protected, meaning that the command must be immediately preceded by a **permit** command if parameters are to be changed.

The command's parameters are:

- **state** [**= on** | **off**] determines whether the channels specified in the **channels** option will be simulated (**on**), or report true measured data (**off**). The setting is persistent; it will remain in force from one deployment to the next. The default setting as shipped from the Factory is **off**.
- **period** [= <*milliseconds*>] is the period in milliseconds of one full cycle of simulated values. The default value is 3600000ms, corresponding to 1 hour.
- **channels** [= <*channel_list*>] is an option available only in firmware versions 1.150 or later, and it specifies a list of the channels that will be simulated. Each channel may be specified either by its index or by its label, but when the channel list is reported indices are always used. The channel specifiers in the list are separated by a pipe character ('|') with no spaces. The order in which channels are specified does not matter.
  Derived channels cannot be simulated; only measured channels may be selected. Derived channels are calculated as usual, whether the underlying measured channels are simulated or not. As shipped from the Factory all measured channels are selected by default.
  There are two reserved keywords that may be used in place of a <*channel_list*>; **all** selects all measured channels for simulation, while **none** can be used to disable simulation for all channels.
  Firmware versions prior to 1.150 always simulate all measured channels if the feature is enabled (**state = on**).

When **state = on**, the measured values from the selected channels 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.

Channel type	Minimum	Maximum	Units
temperature	-5	+35	°C
pressure	+10	+2000*	dbar
conductivity	-1	+85	mS / cm
PAR	-25	+2500	µmol / m^2 / s
turbidity	-25	+2500	NTU
chlorophyll	-2	+150	µg / L
o2 concentration	0	+450	µM
All other types	25% full scale	75% full scale	as appropriate

*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 (whether stored in memory, streamed, or both) and for on-demand samples obtained using the **fetch** 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** command for the indicator as shown below.

```
>> simulation state
<< simulation state = on|off

>> id
<< id mode = SIMULATION, model = RBRconcerto3, version = 1.000, serial = 012345, fwtype
= 104
    (Simulated data is enabled)

>> id
<< id model = RBRconcerto3, version = 1.000, serial = 012345, fwtype = 104
    (Simulated data is disabled)
```

Examples

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

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

>> permit command = simulation
<< permit command = simulation
>> simulation channels = conductivity_00|temperature_00
```

```
<< simulation channels = 1|2
```

Errors

Error E0105 command prohibited while logging
Changes to the **state** or **period** parameters cannot be made while logging is enabled.

Error E0103 protected command, use 'permit command = <command>'
Changes to the **state** or **period** parameters cannot be made without issuing the command **permit command = simulation** immediately beforehand.

Error E0108 invalid argument to command: '<invalid-argument>'
One of the supplied parameter names or values was not valid.

3.5 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.

3.5.1 meminfo

Usage

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

Security

Open.

Description

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

- **dataset** [= <*index*>] is the index of the dataset being queried - see below for details.
- **used** is the number of bytes actually used to store data in this dataset.
- **remaining** is the number of bytes still available for data storage.
- **size** is the maximum total size in bytes of the dataset.
- **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:

- the deployment header is in dataset 2
- the sample data is in dataset 1
- events are in dataset 0
- the sample data generated by postprocessing is in dataset 4

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.

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.

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 invalid argument to command: '<invalid-argument>'
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**.

3.5.2 memclear

Usage

```
>> memclear
```

Security

Unsafe and protected.

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

Errors

Error E0103 protected command, use 'permit command = <command>'
permit command = memclear must immediately precede the command to clear the memory.

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

3.5.3 memformat

Usage

```
>> memformat [ type | newtype | availabletypes ]
```

Security

Unsafe.

Description

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

- **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**.
- **newtype** [= <*type*>] requests or sets the format of the data which will be used during a future deployment; the value cannotbe modified while logging is in progress.
  - **rawbin00**
  - **calbin00**
- **availabletypes** requests a list of the data storage formats available for this logger.

If no argument is given to the command, all arguments are 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.

```
>> memformat availabletypes
<< memformat availabletypes = rawbin00|calbin00
```

```
>> memformat
<< memformat type = rawbin00, newtype = rawbin00, availabletypes = rawbin00|calbin00
>> permit command = memclear
<< permit command = memclear
>> memclear
<< memclear used = 0
>> memformat type
<< memformat type = none
```

```
>> memformat newtype = calbin00
<< memformat newtype = calbin00
>> memformat type
<< memformat type = none
```

Errors

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

Error E0108 invalid argument to command: '<invalid-argument>'
The supplied argument was not recognized.

3.5.4 readdata

Usage

```
>> readdata [ dataset | size | offset ]
```

Security

Open.

Description

This command allows you to read data from the datasets.

- **dataset** [= <*index*>], <*index*> is the dataset index to be read.
- **size** [= <*value*>], <*value*> is the amount of data to be read, if not set, readdata will use the last size used.
- **offset** [= <*value*>], <*value*> is the dataset index to be read, if not set, readdata will use the offset corresponding the offset of the last byte read +1.

The command replies as any other command by confirming the key values pairs dataset, size and offset effectively reported followed by a <CR><LF>. Then send a binary stream composed of:

1. <*data*> is the actual binary data, delimited at the start by the <CR><LF> and bounded in length by size parameter returned.
2. <*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 and using 0xFFFF as a seed value. 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.

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:

- The deployment header is in dataset 2
- The sample data is in dataset 1
- Events are in dataset 0

The **readdata** 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 **readdata** 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

```
>> readdata dataset = 1, size = 1000, offset=0
<< readdata dataset = 1, size = 1000, offset = 0<cr><lf><bytes[0..999]-of-data><crc>
```

```
>> readdata dataset = 1, size = 1000, offset = 1000
<< readdata dataset = 1, size = 1000, offset = 1000<cr><lf><bytes[1000..1999]-of-
```

```
data><crc>
```

```
>> readdata dataset = 1, size = 1000, offset = 2000
<< readdata dataset = 1, size = 12, offset = 2000<cr><lf><bytes[2000..2011]-of-
data><crc>
```

Errors

Error E0108 invalid argument to command: '<invalid-argument>'
The supplied argument was not recognized.

3.5.5 postprocessing

Usage

```
>> postprocessing [ channels | mode | status | command | dc_alpha | dc_tau |
dc_tdelay | dc_ctcoeff ]
```

Security

None.

Description

The postprocessing feature enables the logger to perform various post-processing operations on data measured and stored by the logger. It operates on data stored in EasyParse format in dataset-1 and writes EasyParse format into dataset-4 based on the channels and parameters requested. The postprocessing can occur as soon as there is an EasyParse dataset available (even if logging is in progress). If the instrument is powered cycle while performing the postprocessing, it will continue the postprocessing after power cycling.

This feature was introduced in firmware version 1.136. Firmwares between 1.102 and 1.135 carry a different implementation of the postprocessing command.

- **channels [=** ***listofchannelsstats***] is the list of channels statistics to be obtained. Items in the list are separated by the character | and follow the format <***aggregatefunction***>(<***channel***>), where**<*channel*>**is a channel label and**<*aggregatefunction*>**is either:
  - **mean**, average value in the bin
  - **std**, standard deviation of the bin
  - **count**, number of samples in the bin

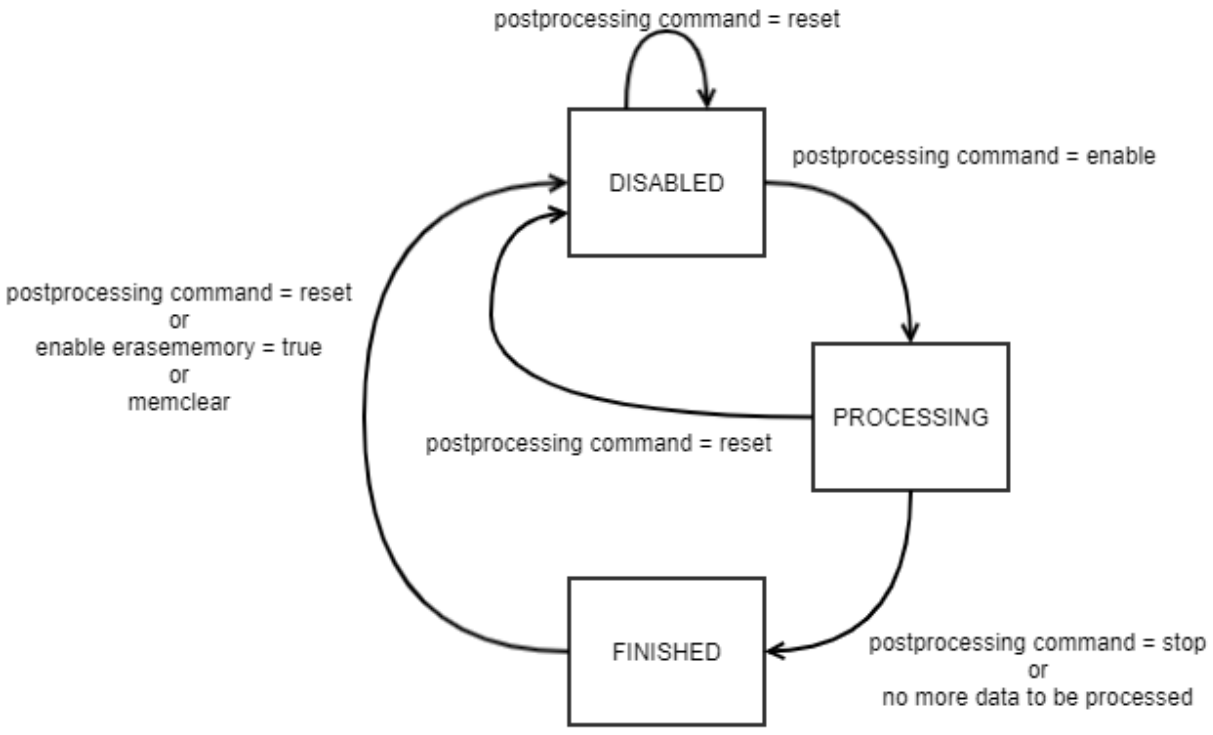
A maximum of 24 channels is supported. The generated dataset-4 channels order will follow the same order as <***listofchannelsstats***>.

- **mode [= continous|regimes] Select the mode of operation for the postprocessing.**
  - When the mode is **regimes**, it indicates the postprocessing will use settings from **postprocessing_regime** and **postprocessing_regimes** commands to process the data. Behaviour mimics the real-time **regime**/**regimes** command.
  - When the mode is **continuous**, all the data is processed without performing any binning. This is used for performing dynamic correction while maintaining the same sampling rate

- **status** is a read-only parameter that returns the current state of the finite state machine for the instrument's post-processing function. Possible statuses are:
  - **disabled**, indicating that the postprocessing is waiting to be enabled. Data stored in the postprocessing dataset (if any) corresponds to the previous postprocessing parameters.
  - **processing**, indicating that a postprocessing job is ongoing. Data stored in the postprocessing dataset corresponds to the current postprocessing configuration but might be incomplete.
  - **finished**, indicating that a postprocessing job has been completed. Data stored in the postprocessing dataset corresponds to the current postprocessing parameters.
- **command** = <*command*> is a write-only parameter that controls the execution of the postprocessing job. The **status** parameter is reported upon issuing a command. <*command*> can be one of:
  - **enable** starts the post-processing job (only if the status is disabled)
  - **stop** stops the post-processing job (indicate completion)
  - **reset** return back to the disabled state (doesn't clear the dataset)

The command **reset** will not discard the dataset. However, if the memory is cleared (either via **memclear** or via **enable erasememory = true**), dataset-4 will be discarded once the memory is erased: the dataset is always associated with the relevant dataset-1. Changing postprocessing parameters is not allowed while **status** is **processing** or **finished**. In order to change any parameters, a **reset** needs to be issued to go back to the **disabled** status.

> The **finished** status indicates previously processed data and current postprocessing parameters are the ones used to process the dataset.



Postprocessing allows calculating dynamic correction for salinity and marine temperature channels on RBRargo C.T.D. A special suffix (**_dyn_corr**) is added to the channels label to expose their dynamically corrected version:

- salinity_00_dyn_corr
- temperature_00_dyn_corr

This method is not recommended anymore as the algorithm used assumes a constant ascent rate and firmware 1.145 provides a salinity channel with dynamic correction applied based on the instantaneous ascent speed.

- **dc_tau** [= <*value*>] is the C-T lag for dynamic correction of the salinity.
- **dc_alpha**[= <*value*>] is the magnitude of short-term thermal mass correction for dynamic correction of the salinity.
- **dc_tdelay**[= <*value*>] is a dynamic correction parameter for the time lag correction (in seconds) between marine temperature and conductivity cell temperature
- **dc_ctcoeff**[= <*value*>] is the magnitude of long-term thermal mass correction for dynamic correction of the salinity.

Examples

```
>> postprocessing
<< postprocessing status = finished, mode = continuous, channels =
mean(conductivity_00)|mean(temperature_00)|mean(pressure_00)|mean(salinity_00_dyn_corr)|
mean(temperature_00_dyn_corr)
>> postprocessing all
<< postprocessing status = finished, mode = continuous, channels =
mean(conductivity_00)|mean(temperature_00)|mean(pressure_00)|mean(salinity_00_dyn_corr)|
mean(temperature_00_dyn_corr), dc_alpha = 0.1300, dc_tau = 5.9000, dc_tdelay = 0.3500,
dc_ctcoeff = 1.0200e-00
```

```
>> postprocessing status
<< postprocessing status = disabled
>> postprocessing command = enable
<< postprocessing status = processing
...[ postprocessing active ] ...
>> postprocessing status
<< postprocessing status = finished
```

```
>> postprocessing dc_tau = 7.00
<< postprocessing dc_tau = 7.00
>> postprocessing dc_alpha
<< postprocessing dc_alpha = 0.08
```

Errors

Error E0108 invalid argument to command

The command was given with an argument which is unrecognized or has an invalid value; for example

Error E0425 invalid settings

Post-processing cannot start because some settings are inconsistent or invalid.

Error E0426 postprocessing already active

Parameters cannot be changed while the post-processing status is not idle.

Error E0427 wrong memory format
The current dataset is not stored using the calbin00 memory format.

Error E0428 postprocessing reference channel not available
One of the supporting channels required is not available in the current dataset.

3.5.6 postprocessing_regimes

Usage

```
>> postprocessing_regimes [ direction | count | reference ]
```

Security

None.

Description

Sets and returns regimes parameters for use with **postprocessing** command. These are only used if the regimes postprocessing mode is in use. This is intended for use when the instrument is integrated into moving platforms.

Changing postprocessing_regimes parameters is not allowed while **postprocessing** **status** is **processing** or **finished**.

- **direction** [= <*direction*>]
  - **ascending** indicates that the instrument is intended to ascend in the water column.
  - **descending** indicates that the instrument is intended to descend in the water column.
- **count** [= <*countvalue*>] indicates the number of regimes that are set. The minimum number is 1 and the maximum number of regimes is 3.
- **reference** [= <*reference*>]
  - **absolute** indicates that the absolute pressure is used as reference for the determination of the current regime and bin.
  - **seapressure** indicates that the sea pressure is used as reference for the determination of the current regime and bin.

Examples

```
>> postprocessing_regimes
<< postprocessing_regimes direction = descending, count = 3, reference = absolute
>> postprocessing_regimes direction = ascending
<< postprocessing_regimes direction = ascending
```

Errors

Error E0108 invalid argument to command
The command was given with an argument which is unrecognized or has an invalid value; for example

Error E0426 postprocessing already active
Parameters cannot be changed while the post-processing status is not idle.

3.5.7 postprocessing_regime

Usage

```
>> postprocessing_regime <index> [ boundary | binsize ]
```

Security

None.

Description

Returns information about the specified *<index>* regime for **postprocessing**. The first regime has an *<index>* of 1. <index> should be lower or equal to the number of regimes reported by the **postprocessing_regimes** command at the time of issuing the command. If the direction (**postprocessing_regimes** command) is set to descending, the first regime corresponds to the regime the closest to the surface. If the direction is set to ascending, the first regime is the closest to the seabed. Depending on how is set the reference (**postprocessing_regimes** command), the logger use the absolute pressure or the sea pressure to determine the current regime and the current bin.

Changing postprocessing_regime parameters is not allowed while **postprocessing** **status** is **processing** or **finished**.

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

- **boundary** [= *<firstboundaryvalue>*] specifies the transition from one regime to the next, and is interpreted as the *first* boundary in a region (lower bound if ascending, upper bound if descending). The units are in dbar. The minimum precision is 1 dbar, and the value should be between 0 dbar and 65535 dbar
- **binsize** [= *<countvalue>*] specifies the size (in dbar) used for each averaged bin. The minimum precision is 0.1dbar, and the value should be between 0.1 dbar and 6553.5 dbar.

Example

```
>> postprocessing_regimes
<< postprocessing_regimes direction = descending, count = 3, reference = absolute
>> postprocessing_regime 1
<< postprocessing_regime 1 boundary = 2000, binsize = 100.0
>> postprocessing_regime 2
<< postprocessing_regime 2 boundary = 4000, binsize = 10.0
>> postprocessing_regime 3
<< postprocessing_regime 3 boundary = 5000, binsize = 20.0
```

Errors

Error E0108 invalid argument to command
The command was given with an argument which is unrecognized or has an invalid value; for example

Error E0426 postprocessing already active
Parameters cannot be changed while the post-processing status is not idle.

3.6 Configuration information and calibration

3.6.1 channels

Usage

```
>> channels [ count | on | settlingtime | readtime | minperiod ]
```

Security

Open.

Description

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

- **count** is simply the number of channels installed and configured in the logger.
- **on** gives the number of active channels, which excludes any channels turned **off** by the user: see the **status** argument to the **channel** command.
- **settlingtime** 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 settling time is determined by the longest of all the individual *active* channel delays; channels which are turned **off** do not contribute.
- **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 settling time has passed. This overall readtime is determined by the longest value of all the individual *active* channels; any which are turned **off** do not contribute.
  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.
- **minperiod** is the minimum sampling period in milliseconds with the currently active channels. This takes account of the current overall values for the settling and read times, 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, settlingtime = 160, readtime = 150, minperiod = 1000
```

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

Errors

Error E0108 invalid argument to command: '<invalid-argument>'
An unrecognized argument was given with the command.

Error E0505 no channels configured
Indicates a serious fault with the logger; please contact RBR Ltd for help.

3.6.2 channel

Usage

```
>> channel <indexorlabel> [ type | module | status | settlingtime | readtime |
equation | userunits | gain | availablegains | derived | label | index ]
```

Security

Unsafe.

Description

Returns information about the channel at the specified <*indexorlabel*>. Channels can be identified either by an index (an integer) or by their label (a meaningful string, for example temperature_00 or conductivity_00). The first channel has an index of 1.

A special value of **allindices** or **alllabels** may be given for <*indexorlabel*> causing the requested parameters to be reported for all channels. The output for each channel is terminated by ||, 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. All parameters might be obtained using the **all** in lieu of parameter names.

- **type** is a short, pre-defined 'generic' name for the installed channel; for example:
  - **temp09** RBR*duo*^{3} temperature
  - **pres19** RBR*duo*^{3} pressure
  - **cond05** RBR*concerto*^{3} marine conductivity,
    and so on. RBR Ltd continually adds support for more sensor types and variants; the Section Supported Channel Types contains a complete listing of channel types available at the time of writing this document.
- **module** is the internal address to which this channel responds; it is normally of no interest to end users.
- **status** [= <*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.
  - **off**: the channel is not sampled, no data will be stored in memory or streamed for this channel.
- **settlingtime** is the minimum power-on settling delay in milliseconds required by this channel, taking into account both the sensor and the interface electronics.
- **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.

- **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 for details of all supported equations.
  - **tmp** temperature
  - **lin** linear
  - **qad** quadratic polynomial
  - **cub** cubic polynomial
- **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.
- **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 cannot be modified: it is for information only.
- **gain** reports the gain setting currently in use by the channel referenced by <*indexorlabel*>. In addition to one of the fixed values from the list reported by the **availablegains** 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 **availablegains** 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.
- **availablegains** reports the gain settings supported by the sensor at channel <*indexorlabel*>. 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 <*indexorlabel*> does not support multiple gain settings, the response is **none**.

The **availablegains** 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.

- **label** is a short text string without white spaces, commas, equal symbols, or special characters, describing the physical parameter measured (example: **temperature_00**). It is reported when the channel requested is referred by its index.
- **index** is the index of the channel. It is reported when the channel requested is referred by its label.

Examples

```
>> channel 1
<< channel 1 type = temp09, module = 6, status = on, settlingtime = 50, readtime = 260,
equation = tmp, userunits = C, derived = off, label = temperature_00
```

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

```
>> channel allindices type
<< channel 1 type = temp09 || channel 2 type = pres19
>> channel alllabels type
```

```
<< channel temperature_00 type = temp09 || channel pressure_00 type = pres19
```

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

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

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

```
>> channel 1 all
<< channel 1 type = temp14, module = 1, status = on, settlingtime = 50, readtime = 260,
equation = tmp, userunits = C, gain = none, availablegains = none, derived = off, label
= temperature_00
```

Errors

Error E0105 command prohibited while logging

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

Error E0107 expected argument missing

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

Error E0108 invalid argument to command: '<invalid-argument>'

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 command failed

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 no channels configured

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 item is not configured

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

Further **status** parameter details

Example use-cases

RBR*concerto*^{3} C.T.D; real-time 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 (e.g., 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.

RBR*concerto*^{3} C.T.D; real-time 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".

Storage implications

For storage of data in standard format (rawbin00):

- Only raw, uncorrected, measured channels are stored.
- Corrected or derived channels are never stored.
- 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.
- 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):

- All channels which are **on** are stored, including any derived channels.
- 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**.

3.6.3 settings

Usage

```
>> settings [ fetchpoweroffdelay | sensorpoweralwayson | castdetection |
inputtimeout | speccondtempco | altitude | temperature | pressure | atmosphere |
density | salinity | avgsoundspeed ]
```

Security

Unsafe.

Description

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

- **fetchpoweroffdelay** [= <*timeoutinmilliseconds*>] 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.

- **sensorpoweralwayson** [= <*state*>] 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**.
- **castdetection** [= <*state*>] 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**.
- **inputtimeout** [= <*timeoutinmilliseconds*>] 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 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.
- **speccondtempco** [= <*value*>] 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.
- **altitude** [= <*value*>] is the height above the seabed in metres 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.
- **temperature**, **pressure**, **atmosphere**, **density, salinity**, **avgsoundspeed** [= <*value*>]: 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.
  - temperature in °C, default value 15.0
  - absolute pressure in dbar, default value 10.132501 (1 standard atmosphere)
  - atmospheric pressure in dbar, default value 10.132501
  - water density in g/cm3, default value 1.026021
  - salinity in PSU, default value 35
  - avgsoundspeed in m/s, default value 1506.8

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 command prohibited while logging
Parameters may not be modified while logging is in progress.

Error E0108 invalid argument to command: '<invalid-argument>'
The supplied argument was not valid; examples include: invalid parameter name and improperly specified value.

3.6.4 calibration

Usage

```
>> calibration <indexorlabel> [ datetime | c0 | … | cn | x0 | … | xn | n0 | … | xn ]
```

Security

Unsafe and protected.

Description

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

A special value of **allindices** or **alllabels** may be given for <*indexorlabel*> causing the requested parameters to be reported for all channels. The output for each channel is terminated by ||, except for the last channel which is terminated by a <**cr**><**lf**> pair as normal.

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**.
All parameters might be obtained using the **all** keyword in lieu of parameter names. 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 (e.g., **c3** for a linear sensor) may result in either an error message, or a response such as **c3 = n/a**.

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

- Some parameters are read only; **n0…** .
- **datetime =** <*YYYYMMDDhhmmss*> must accompany any changes to coefficient values, so that the reported date and time reflects the most recent change.

- a single **calibration** command can set coefficient values in only one of the groups at a time; **c0**…, **x0**…, Coefficients from different groups cannotbe mixed in a single command when setting.

Descriptions of the individual parameters are given below.

- **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.
- **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.
- **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.
- **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 cannotbe 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 for a complete list of the equations which the logger uses, and for further discussion of cross channel dependencies.

Examples

```
>> calibration 1
<< calibration 1 label = voltage_01, datetime = 20171218175005, c0 = 9.9876543e+000, c1
= 7.5642301e+000

>> calibration voltage_01
<< calibration voltage_01 index = 1, datetime = 20171218175005, c0 = 9.9876543e+000, c1
= 7.5642301e+000
```

Querying the calibration for a single channel by index and by label.

```
>> calibration allindices
<< calibration 1 label = voltage_01, datetime = 20171203134201, c0 = 9.9873456e+000, c1
= 7.5640000e+000 || calibration 2 label = voltage_02, datetime = 20171203134201, c0 =
```

```
9.9873456e+000, c1 = 7.5640000e+000

>> calibration alllabels
<< calibration voltage_01 index = 1, datetime = 20171203134201, c0 = 9.9873456e+000, c1
= 7.5640000e+000 || calibration voltage_02 index = 2, datetime = 20171203134201, c0 =
9.9873456e+000, c1 = 7.5640000e+000
```

Querying the calibration for all channels by indices and by labels.

```
>> permit command = calibration
<< permit command = calibration
>> calibration 1 datetime = 20171203134201, c0 = 9.9873456, c1 = 7.564
<< calibration 1 datetime = 20171203134201, c0 = 9.9873456e+000, c1 = 7.5640000e+000
```

Setting the calibration for a channel.

Errors

Error E0103 protected command, use 'permit command = <command>'
permit command = calibration must immediately precede the command if setting coefficients.

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

Error E0107 expected argument missing
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 invalid argument to command: '<invalid-argument>'

The supplied argument was not valid; examples include:

- *<index|label>* out of range; channels are numbered from 1 to N; zero is not valid, or no channel exists with this label.
- improperly formatted or invalid *<datetime>* argument.
- invalid coefficient name.
- improperly specified value for a coefficient.

Error E0501 item is not configured
There is a problem with the configuration of the specified channel; please contact RBR Ltd for help.

Error E0505 no channels configured
Indicates a serious fault with the logger; please contact RBR Ltd for help.

3.6.5 sensor

Usage

```
>> sensor <indexorlabel> [ param1 | … | paramn]
```

Security

Unsafe.

Description

Returns information about the sensor attached to the channel at the specified *<indexorlabel>* (see **channel**)

A special value of **allindices** or **alllabels** may be given for <*indexorlabel*> causing the requested parameters to be reported for all channels. The output for each channel is terminated by ||, except for the last channel which is terminated by a <**cr**><**lf**> pair as normal.
All parameters might be obtained using the **all** keyword in lieu of parameter names but this one is optional.

The purpose of this command 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** command is used; a good example would be the **settlingtime**, 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 cannot 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.

Examples

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

Retrieve all information pertaining to channel 3. In this example, only the "serial" parameter is available.

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

Request a specific parameter.

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

Change the value of a parameter.

```
>> sensor 4
<< sensor 4
```

No sensor information is available for channel 4.

```
>> sensor 4 serial
<< sensor 4 serial = n/a
```

The "serial" parameter is unavailable for channel 4.

```
>> sensor allindices serial
<< sensor 1 serial = n/a || sensor 2 serial = n/a || sensor 3 serial = 129837 || sensor
4 serial = n/a
```

Errors

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

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

Error E0108 invalid argument to command: '<invalid-argument>'
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 command failed
There was a problem reading or modifying some data for the specified parameter and/or channel. Please contact RBR Ltd for help.

Error E0505 no channels configured
There is a serious problem with the logger's configuration. Please contact RBR Ltd for help.

Error E0501 item is not configured
There is a problem with the configuration of the specified channel; please contact RBR Ltd for help.

3.6.6 valve

Usage

```
>> valve [ id | scheduled | powerondelay | poweroffdelay | interval | duration |
command | timetoepisode | startimmediate | episodelog | operationcount |
scheduletype]
```

Security

Unsafe.

Description

This is a specialized command that has previously appeared only in specific firmware versions (1.119, 1.118, 1.115) used in BPR|zero logger systems. This documentation now describes the command as implemented in all firmware versions 1.132 or later; there will be some differences from the earlier implementations.

The command returns information about the valve controller used in BPR|zero logger systems, and also allows some control over it.

- **id** reports (read-only) the type of valve configured to be used with the instrument.
- **scheduled** [**= true** | **false**] reports or sets whether or not the valve schedule is enabled for use. The valve can always be operated by user commands at any time, provided adequate external power is available.
  - **true**: this is the factory default; the valve schedule is enabled for the current or next deployment
  - **false**: the valve schedule is not enabled.

The valve schedule operates only when the instrument is enabled for logging. It will not operate before the logging start-time, after the logging end-time, or in any other situation where logging is disabled. Simply setting **scheduled = true** will *not* immediately start the valve schedule if logging is not enabled.

- **powerondelay** reports (read-only) the startup time for the valve in milliseconds.
- **poweroffdelay** reports (read-only) the shut down time for the valve in milliseconds.
- **interval** [= <*milliseconds*>] reports or sets the interval between valve episodes in milliseconds, for a **uniform** schedule (see **scheduletype** below).
  - The minimum interval is 120000, or 2 minutes.
  - For firmware versions prior to 1.142, the maximum interval is 3888000000, corresponding to 45 days. For firmware versions 1.142 or later, the limit has been extended to the equivalent of 45000 days, which in practice is no limit at all.
  - Only values rounded to the minute are valid, so the number must be a multiple of 60000.
  - The interval must be greater than the duration.
  - The factory default value is 3600000ms, or 1 hour.
- **duration** [= <*milliseconds*>] reports or sets the duration of a valve episode in milliseconds.
  - The minimum duration is 60000 (1 minute).
  - The maximum duration is 86400000 milliseconds, which corresponds to 1 day.
  - The duration must be less than the interval period.
  - Only values rounded to the minute are valid, so the number must be a multiple of 60000.
  - The factory default value is 60000ms, or 1 minute.
- **command** [= <*command*>] sends a particular command to the valve; in the case of a BPR|zero instrument, <*command*> can be one of the following options. In all cases, the system may take up to 3000 milliseconds (nominally) to respond with the valve position or return an error message, in addition to the **powerondelay** and **poweroffdelay** times.
  - **setpositionM**, moves the valve to Position M (Marine).
  - **setpositionA**, moves the valve to Position A (Atmospheric).
  - **getposition**, reports the current position or status of the valve. In the event of a fault, the status may indicate the error.
    - **positionM**
    - **positionA**
    - <*specific_error_message*>

Moving the valve, or activating it to request its position, requires a significant amount of power which cannot be provided by the instrument's internal batteries. Before these operations, the instrument checks that the external power is available and meets the minimum specified voltage threshold (approximately 9V). If it does not, the response to these commands is an error message of the form **valve status = powerfail**.

When possible, the instrument remembers the valve position, and so **getposition** may respond with a valid position even when there is no power. The reasoning is that if there is no power, the valve cannot have moved since its position was last known.

- **timetoepisode** reports (read-only) the time remaining until the start of the next scheduled episode. The value is reported in milliseconds, but only to a resolution of one second. If the valve schedule is not enabled, the response is **valve timetoepisode = n/a**. During an active schedule, the value reported can range from 1000 up to **interval** milliseconds. If the schedule is enabled but logging has not yet started, the reported value will reflect the time until the start of the first episode, correctly accounting for the setting of the **startimmediate** parameter (see below).

This reported value may be longer than the **interval**, of course.

- **startimmediate** [**= true** | **false**] reports or sets the point in a **uniform** schedule (see **scheduletype** below) at which the first valve episode will occur:
  - **true**: this is the factory default; a valve episode will occur immediately at the start of the schedule, or following a reset.
  - **false**: the first valve episode will be delayed by one **interval** after the start of the schedule or a reset.
- **operationcount** reports (read-only) the number of valve movements that have been performed *or attempted* since the instrument was configured in the Factory. This can be used to determine whether the valve is approaching its specified number of guaranteed operations. To be conservative, attempted movements resulting in some form of error are also included in the count, as there is no way of telling whether these contribute more or less to general wear and tear than successful movements.
- **episodelog** [**= on** | **off**] can be used to enable, disable, or report whether valve events will be recorded in the instrument's memory when data logging is enabled. Events occurring outside the data logging deployment are never recorded. When **episodelog = on**, an event will be recorded in memory during the deployment each time an attempt is made to change the state of the valve, whether successful or not. This applies to both scheduled events and manually commanded events. Each event contains information to show which position the valve was moved to, or whether some error occurred in the attempt. The default setting of this parameter is **episodelog = on**, and it is recommended to keep this setting in normal circumstances so that a record of valve activity is preserved. If desired, it can be turned off for testing or other purposes.
- **scheduletype** [**= uniform** | **segmented**] is an option available in firmware versions 1.142 or later. The default setting is **uniform**, in which the occurrence of episodes within the schedule is governed by the **interval**, **duration**, and **startimmediate** parameters discussed above (this also applies to all firmware versions prior to 1.142 that support the valve controller). Alternatively, selecting **segmented** allows the schedule of valve episodes to be divided into a series of segments. Each segment effectively has its own "start time", and an interval between episodes that can be specified independently of other segments. Refer to the **valvesegments** and **valvesegment** commands for further details. For a segmented schedule, the **startimmediate** and **interval** parameters discussed above do not apply; currently however, the **duration** parameter always applies to the entire schedule, whether **segmented** or **uniform**.

Examples

```
>> valve id
<< valve id = valveA
```

```
>> valve scheduled
<< valve scheduled = false
>> valve scheduled = true
<< valve scheduled = true
```

```
>> valve powerondelay, poweroffdelay
<< valve powerondelay = 1300, poweroffdelay = 50
```

```
>> valve command = getposition
<< valve status = positionM

>> valve command = setpositionA
<< valve status = positionA
```

```
>> valve command = getposition
<< valve  status = positionA
```

```
>> valve command = setpositionA
<< valve status = powerfail
```

There is not adequate external power available to move the valve.

```
>> valve interval, duration
<< valve interval = 3600000, duration = 300000
```

Errors

Error E0114 feature not supported by hardware
An attempt was made to use the command with a logger that does not have the BPR|zero valve controller installed.

Error E0105 command prohibited while logging
Some settings may not be modified while logging is in progress; reading them is permitted.

Error E0107 expected argument missing
The attempted operation requires an additional argument that was not specified with the command.

Error E0108 invalid argument to command
The argument supplied is not recognized as a valid argument, or an attempt was made to set the value of a read-only parameter.

Error E0111 command failed
An unidentified problem prevented the operation from being completed.

Error E0702 no devices configured
The configuration of the valve controller interface on this instrument is not correct.

Error E0703 device schedule inconsistent
The interval period is not greater than the duration.

Error E0704 device is not enabled
The attempted operation could not be performed because the device is disabled.

Error E0705 multiple operations not supported: '*<extra_operation>*'
Some valve operations cannot be combined in a single command; try sending them separately.

Error E0701 device error:
An internal error occurred. Please contact RBR Ltd for help.

3.6.7 valvesegments

Usage

```
>> valvesegments [ count ]
```

Security

Unsafe.

Description

This is a specialized command available in firmware versions 1.142 or later. It is used in BPR|zero logger systems to help implement a more flexible schedule of valve episodes than was previously available.

If desired, the schedule of valve episodes can now be divided into a series of segments. Each segment effectively has its own "start time", and an interval between episodes that can be specified independently of other segments. This command sets or reports the number of segments that will be used during the schedule.

- **count** [**= 1...4**] reports or sets the number of segments in the valve schedule. The minimum number of segments is 1, and the maximum is 4.

Refer to the **valvesegment** command for information on accessing the parameters for each segment.

Examples

```
>> valvesegments
<< valvesegments count = 2
>> valvesegments count = 3
<< valvesegments count = 3
```

Errors

Error E0114 feature not supported by hardware
An attempt was made to use the command with a logger that does not have the BPR|zero valve controller installed.

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

Error E0108 invalid argument to command
The argument supplied is out of range, or is not recognized as a valid argument.

Error E0111 command failed
An unidentified problem prevented the operation from being completed.

Error E0702 no devices configured
The configuration of the valve controller interface on this instrument is not correct.

Error E0703 device schedule inconsistent
Parameters in one of the enabled segments are not consistent.

3.6.8 valvesegment

Usage

```
>> valvesegment <segment_number> [ delay | interval ]
```

Security

Unsafe.

Description

This is a specialized command available in firmware versions 1.142 or later. It is used in BPR|zero logger systems to help implement a more flexible schedule of valve episodes than was previously available.

If desired, the schedule of valve episodes can now be divided into a series of segments. Each segment effectively has its own "start time", specified by the **delay**, and an **interval** between episodes that can be specified independently of other segments. This command sets or reports the parameters associated with the requested segment <*segment_number*>.

- <*segment_number*> is a required parameter giving the index of the segment to be accessed. The first segment has an index of 1; this is the minimum value accepted. The maximum acceptable value of the segment index is equal to the segment **count**, accessible using the **valvesegments** command. The keyword **all** in place of a <*segment_number*> causes the parameters for all valid segments to be reported, segments being separated by a pair of pipe (vertical bar) characters, '||'. Segment 1 is at the start of the schedule/deployment and is required; the remaining segments are optional, and may be included or omitted by using the **valvesegments count** command. If further segments are included, then they follow Segment 1 in order as the schedule/deployment progresses; it is not possible to 'skip' a segment.
- **delay =** <*delay_ms*>. This is the segment delay, measured in milliseconds from the start of the *previous* segment; for Segment 1, it is measured from the first sample of the deployment, After this delay has expired, the given <*interval_ms*> and the universal **duration** are used for the valve schedule. A delay value of zero is permitted only for Segment 1, so that Segment 1 can start immediately when the schedule/deployment begins; a delay value of zero for any other segment is forbidden.
- **interval =** <*interval_ms*>. The time in milliseconds between the start of consecutive valve episodes, applicable during the specified segment.

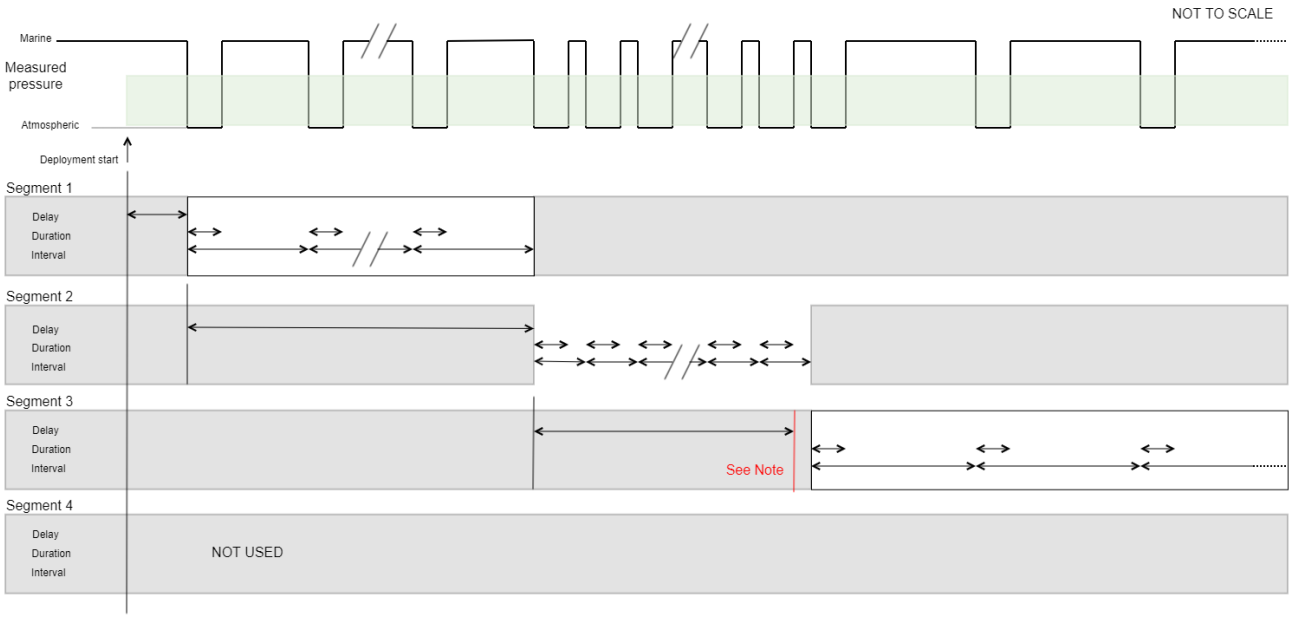
Segment parameters can be read and modified while the **scheduletype** is **uniform**, and values will be retained even if a **uniform** schedule is in use; however, segment parameter settings will not become effective until the **scheduletype** is changed to **segmented**.

When querying or specifying segment parameters, the <*segment_number*> (or **all**) is always required and must be specified first. When setting values, the logger will do some basic error checking, but it is the user's responsibility to ensure that all three time parameters (**delay**, **interval**, and universal **duration**) are consistent with a valid schedule. It is not possible to update parameters for multiple segments at once; each segment must be updated separately. The logger will perform a final check on the consistency of all parameter values before the instrument is enabled.

All time parameters are in milliseconds, to be consistent with other time parameters for the **valve** command and many other logger commands. All values can be specified to a resolution of 1 minute (60000ms). The minimum **delay** for Segment 1 is zero; for other segments the minimum value is 60000 (1 minute). The minimum **interval** is 120000 (2 minutes), since the **interval** must be greater than the **duration**, and the minimum for the universal **duration** is one minute. The maximum value for both <*delay_ms*> and <*interval_ms*> is the equivalent of 45000 days, which in practice is no restriction at all. The maximum **duration** is 86400000, corresponding to 1 day.

As the deployment progresses, the instrument evaluates which segment is applicable at the start of each valve episode, and not continuously between or during episodes. Thus specifying a delay which expires between intervals would mean that the <*new_interval*> value does not take effect immediately, but at the start of the next episode scheduled according to the <*current_interval*>.

The diagram below should help to clarify the relationships between the parameters for different segments.



Segment 3 in the above example illustrates what happens if a delay is specified that does not coincide with the start of an interval. The logger re-evaluates the schedule only at the start of each interval according to the *current* segment parameters, so the parameters for Segment 3 do not take effect until the current interval of Segment 2 has ended.

Examples

```
>> valvesegment 1
<< valvesegment 1 delay = 0, interval = 7200000

>> valvesegment 1 delay = 86400000
<< valvesegment 1 delay = 86400000

>> valvesegment all
<< valvesegment 1 delay = 86400000, interval = 7200000 || valvesegment 2 delay =
604800000, interval = 86400000

>> valvesegment 3 interval = 172800000
<< E0108 invalid argument to command '3'
>> valvesegments
<< valvesegments count = 2
>> valvesegment 2 interval = 172800000
<< valvesegment 2 interval = 172800000
```

Errors

Error E0114 feature not supported by hardware

An attempt was made to use the command with a logger that does not have the BPR|zero valve controller installed.

Error E0105 command prohibited while logging

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

Error E0107 expected argument missing
A required argument to the command was not given; for example, there must always be a <*segment_index*> specifying the segment to access.

Error E0108 invalid argument to command
The argument supplied is out of range, or is not recognized as a valid argument.

Error E0111 command failed
An unidentified problem prevented the operation from being completed.

Error E0702 no devices configured
The configuration of the valve controller interface on this instrument is not correct.

3.6.9 uvled

Usage

```
>> uvled
```

[id | scheduled | powerondelay | poweroffdelay | interval | duration | command | timetoepisode | startimmediate | operatingtime | episodelog]

Security

Unsafe.

Description

USER CAUTION REQUIRED

The UV-LED antifouling device emits ultra-violet radiation in the C-band. Prolonged exposure may cause skin and/or eye irritation, and should be avoided; it is recommended that appropriate shielding and safety glasses are used when operating the instrument. UV radiation is not directly visible to the naked eye, and immersion in water offers no protection.

This is a specialized command available in firmware versions 1.130 or later. It is used to return information about, and to allow some control over, the UV-LED antifouling device used in the RBR*concerto*^{3} CTD|UV data logger, and other instruments equipped with the UV antifouling feature. If the command is sent to an instrument that does not have the UV antifouling feature, the instrument will respond with an error message:

Error E0114 feature not supported by hardware

The following options are available for use with this command.

- **id** reports (read-only) the type of UV-device configured to be used with the instrument.
- **scheduled** [**= true** | **false**] reports or sets whether or not the UV-LED schedule is enabled for use. The UV-LEDs can always be operated by user commands at any time, provided adequate power is available.
  - **true**: the UV-LED schedule is enabled for the current or next deployment.
  - **false**: the UV-LED schedule is not enabled. This is the as-shipped factory default, for safety reasons when shipping.

The UV-LED schedule operates only when the instrument is enabled for logging. It will not operate before the logging start-time, after the logging end-time, or in any other situation where logging is disabled.

Simply setting **scheduled = true** will *not* immediately start the UV-LED schedule if logging is not enabled.

- **powerondelay** reports (read-only) the startup time for the UV-LEDs in milliseconds.
- **poweroffdelay** reports (read-only) the shut down time for the UV-LEDs in milliseconds.
- **interval** [= <*milliseconds*>] reports or sets the interval between UV-LED episodes in milliseconds.
  - Minimum interval is 60000, or 1 minute.
  - The maximum interval is 3888000000, corresponding to 45 days.
  - Only values rounded to the minute are valid, so the number must be a multiple of 60000.
  - The interval must be greater than the duration.
  - The as-shipped factory default value is 600000ms, or 10 minutes.
- **duration** [= <*milliseconds*>] reports or sets the duration of an UV-LED episode in milliseconds.
  - Minimum duration is 1000 (1 second).
  - Maximum is 86400000 milliseconds, which corresponds to 1 day.
  - The duration must be less than the interval period.
  - Only values rounded to the second are valid, so the number must be a multiple of 1000.
  - The factory default value is 1000ms, or 1 second.
- **command** [= <*command*>] sends a particular command to the UV-LED device; <*command*> can be one of the following options. In all cases, the system typically responds within 1 second.
  - **activate**, turns the UV-LEDs ON.
  - **deactivate**, turns the UV-LEDs OFF.
  - **status**, reports the current ON/OFF state of the UV-LEDs.
    - **activated**
    - **deactivated**

For testing, bench trials, and other non-deployment scenarios, it is normally possible to power an RBR*concerto*^{3} from a USB connection alone. However, accessing a device - and in particular turning the UV-LEDs ON - can require a significant amount of power which a USB port may not be able to provide. Before these operations, the instrument checks that there is a source of available power other than USB; if USB is the *only* power source available, the response to these commands is an error message, **uvled status = powerfail**.

- **timetoepisode** reports (read-only) the time remaining until the start of the next scheduled UV-LED episode. The value is reported in milliseconds, but only to a resolution of one second. If the UV-LED schedule is not enabled, the response is **uvled timetoepisode = n/a**. During an active schedule, the value reported can range from 1000 up to **interval** milliseconds. If the schedule is enabled but logging has not yet started, the reported value will reflect the time until the start of the first episode, correctly accounting for the setting of the **startimmediate** parameter (see below). This reported value may be longer than the **interval**, of course.

- **startimmediate** [**= true** | **false**] reports or sets the point in the schedule at which the first UV-LED episode will occur.
  - **true**: this is the factory default; a UV-LED episode will occur at the start of the schedule, or following a reset. Normally, there may be a delay of up to 1 minute, because episodes are constrained to start on a 1-minute boundary according to the instrument's internal clock. However in some situations it is possible for an episode to start immediately the instrument is enabled for sampling.
  - **false**: the first UV-LED episode will be delayed by one **interval** after the start of the schedule or a reset.
- **operatingtime** reports (read-only) in milliseconds the cumulative time for which the UV-LEDs have been energized since configuration at the Factory. The optical output power of the UV-LEDs reduces over time, and this parameter can be used to make decisions about whether the LEDs should be refurbished or replaced.
- **episodelog** [**= on** | **off**] can be used to enable, disable, or report whether UV-LED events will be recorded in the instrument's memory when data logging is enabled. Events occurring outside the data logging deployment are never recorded. When **episodelog = on**, an event will be recorded in memory during the deployment each time an attempt is made to change the state of the UV-LEDs, whether successful or not. This applies to both scheduled events and manually commanded events. Each event contains information to show whether the UV-LEDs were turned on, or turned off, or whether some error occurred in the attempt. The default setting of this parameter is **episodelog = on**, and it is recommended to keep this setting in normal circumstances so that a record of UV-LED activity is preserved. If desired, it can be turned off for test purposes, or at very low sample rates where the number of episode events is comparable to the number of samples.

Examples

```
>> uvled id
<< uvled id = RBR_uvled_00
```

```
>> uvled scheduled
<< uvled scheduled = false
>> uvled scheduled = true
<< uvled scheduled = true
```

```
>> uvled powerondelay, poweroffdelay
<< uvled powerondelay = 10, poweroffdelay = 10
```

```
>> uvled command = status
<< uvled status = deactivated
>> uvled command = activate
<< uvled status = activated
>> uvled command = status
<< uvled  status = activated
```

```
>> uvled command = activate
<< uvled status = powerfail
```

There is no power source available to activate the UV-LEDs; USB alone cannot be used.

```
>> uvled interval, duration
<< uvled interval = 600000, duration = 6000
```

Errors

Error E0114 feature not supported by hardware
An attempt was made to use the command with a logger that does not have the UV antifouling feature installed.

Error E0105 command prohibited while logging
Some settings may not be modified while logging is in progress; in most cases reading them is permitted.

Error E0107 expected argument missing
The attempted operation requires an additional argument that was not specified with the command.

Error E0108 invalid argument to command
The argument supplied is not recognized as a valid argument, or an attempt was made to set the value of a read-only parameter.

Error E0111 command failed
An unidentified problem prevented the operation from being completed.

Error E0702 no devices configured
The configuration of the UV-LED on this instrument is not correct.

Error E0703 device schedule inconsistent
The interval period is not greater than the duration.

Error E0704 device is not enabled
The attempted operation could not be performed because the device is disabled.

Error E0705 multiple operations not supported: '*<extra_operation>*'
Some operations cannot be combined in a single command; try sending them separately.

Error E0701 device error:
An internal error occurred. Please contact RBR Ltd for help.

3.7 Communications

3.7.1 link

Usage

```
>> link [ type ]
```

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, or Wi-Fi connection.

It reports a single parameter:

- **type**, is the communication link in use, either: usb, serial and Wi-Fi.

Examples

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

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

```
>> link
<< link type = wifi
```

Errors

None.

3.7.2 serial

Usage

```
>> serial [ baudrate | mode | availablemodes | availablebaudrates]
```

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** [= <*baudrate*>]: baudrate of the serial link.
- **mode** [= <*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) 5KΩ 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.

- **availablemodes**: report the list of available modes.
- **availablebaudrates**: report the list of available baudrates

Examples

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

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

```
>> serial mode
<< serial mode = rs232
>> serial mode = rs485f
<< serial mode = rs485f
```

```
>> serial availablebaudrates

<< serial availablebaudrates = 115200|19200|9600|4800|2400|1200|230400|460800
```

```
>> serial availablemodes
<< serial availablemodes = rs232|rs485f|uart|uart_idlelow
```

Errors

Error E0108 invalid argument to command: '<invalid-argument>'
The supplied argument was not a recognized parameter name, baud rate value, or mode setting.

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

3.7.3 sleep

Usage

```
>> sleep [ confirmation = true | false ]
```

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). Power saving measures typically include:

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

If used without any parameters, the command generates no error messages, and always succeeds silently; there is no prompt or confirmation message.

Firmware versions 1.116 or later support an optional **confirmation** parameter. If this is used with a value of **true**, a confirmation message is generated, but no prompt.

- **confirmation** = **true** | **false** When a confirmation is asked for using 'confirmation = true', the sleep command will respond with 'sleep status = sleeping' before entering sleep mode.

Examples

```
>> sleep
```

```
>> sleep confirmation = false
```

```
>> sleep confirmation = true

<< sleep status = sleeping
```

Errors

Error E0108 invalid argument to command: '<invalid-argument>'
The supplied argument was not a recognized parameter name.

3.7.4 wifi

Usage

```
>> wifi [ enabled | state| timeout | commandtimeout | baudrate ]
```

Security

Unsafe.

Description

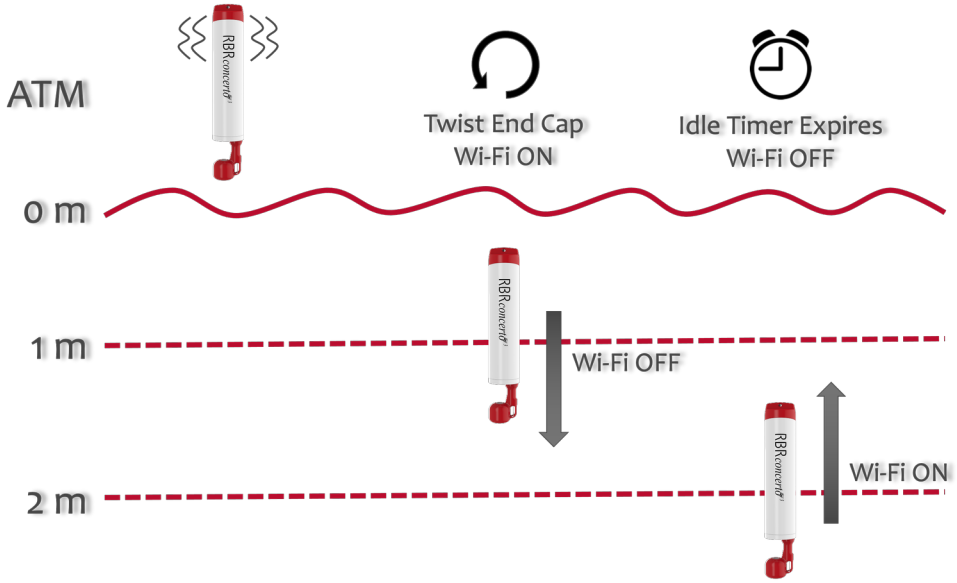
This command allows the user to retrieve information about the Wi-Fi link, and to control some aspects of its behaviour. Please note that some instruments may not have a Wi-Fi link available.

The parameters currently supported are:

- **enabled** [**= true**|**false**], enables (true) or disables (false) Wi-Fi functionality. The logger may support a Wi-Fi link, but

it will not be used if disabled.

- **state** is a read-only parameter which shows whether an enabled Wi-Fi link is **on** (powered up and ready to communicate) or **off** (powered down); a disabled Wi-Fi link will report **n/a** for the **state**.
- **timeout** [= <*timeout*>], specifies how long in seconds an enabled Wi-Fi link will wait for the *first* valid command after powering up. If the timeout expires, the Wi-Fi link is powered down. The timeout can be set to a value from 5 up to 600 seconds; the default value is 60 seconds.
- **commandtimeout** [= <*commandtimeout*>], specifies how long in seconds an enabled Wi-Fi link will wait between commands *after* the first valid one. If the timeout expires, the Wi-Fi link is powered down. The timeout can be set to a value from 5 up to 600 seconds; the default value is 60 seconds.
- **baudrate** is a read-only parameter reporting the speed of an internal connection between the Wi-Fi module and the instrument's CPU. This information may be of use in optimizing data transfer parameters when downloading data from the logger via the Wi-Fi link.



Examples

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

Errors

Error E0108 invalid argument to command: '<invalid-argument>'

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

Error E0105 command prohibited while logging

Parameters may not be modified while logging is in progress; reading the parameter values is permitted.

Error E0109 feature not available

The instrument does not support a Wi-Fi link.

Error E0114 feature not supported by hardware
The instrument is not capable of supporting a Wi-Fi link.

3.8 Other Information

3.8.1 id

Usage

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

Security

Open.

Description

This is a read-only command which report basic information about the instrument:

- **model**, model name of the instrument.
- **version**, firmware version.
- **serial**, serial number is always reported using six digits, padded with leading zeroes if necessary.
- **fwtype**, always report 104 for RBR*virtuoso*^{3}, RBR*duo*^{3}, RBR*concerto*^{3}, and RBR*maestro*^{3} models and all the L3 platform based models. The **fwtype** parameter is used to identify different products among the range of RBR's dataloggers.
- **mode**, only reported when simulation mode is enabled, with the value **SIMULATION**

Examples

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

```
>> id
<< id model = RBRoem, version = 1.000, serial = 050032, fwtype = 104
```

```
>> simulation
<< simulation state = on, period = 3000
>> id
<< id mode = SIMULATION, model = RBRconcerto3, version = 1.000, serial = 012345, fwtype
= 104
```

Refer to the command **simulation** for more details.

Errors

None.

3.8.2 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 invalid command '<unknown-command-name>'
Help was requested for an unknown command.

3.8.3 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 parameters reported are:

- **pcb**, is a letter such as 'C', 'F', etc., which represents the revision level of the main PCB (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 = J, cpu = 5659A, bsl = A
```

Errors

None.

3.8.4 power

Usage

```
>> power [ source | int | ext | reg ]
```

Security

Open.

Description

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

- **source** is one of the following names:
  - **usb** the logger is drawing power from the USB connection.
  - **int** the logger is drawing power from its internal battery.
  - **ext** the logger is drawing power from an external power source.
- **int** reports the measured voltage of a standard logger's internal battery; for a short logger, reports **n/a**.
- **ext** reports the measured voltage of any external power source attached.
- **reg** reports the measured voltage of a short logger's internal voltage regulator; for a standard logger, reports **n/a**.

Support for short instruments starts with firmware version 1.080; earlier firmware versions do not report the **reg** parameter. A short instrument is one which takes only four AA cells instead of the standard eight. In a short instrument, a direct measurement of the internal battery cannot be made, and the output of an internal voltage regulator is reported instead. Standard instruments do not have an equivalent regulator, but can monitor the battery voltage directly.

Examples

```
>> power
<< power source = usb, int = 12.40, ext =  0.00, reg = n/a
```

```
>> power int
<< power int = 12.39
```

Errors

Error E0108 invalid argument to command: '<invalid-argument>'
The supplied argument was not a recognized parameter name.

Error E0111 command failed
A requested parameter could not be measured because of an A/D converter failure.

3.8.5 powerinternal

Usage

```
>> powerinternal [ batterytype | capacity | used ]
```

Security

Open.

Description

Allows various parameters to be reported or set for the current deployment.

- **batterytype** [= <*batterytype*>], has a value corresponding to a chemical description of the various battery types supported; see the list below. The special name **none** indicates that no battery considered to be internal to the logger is present, and it will run exclusively from an external power source.

  The **batterytype** may be set prior to a deployment; it cannot be changed while a deployment is in progress. If proper estimates are required for battery capacity used and deployment life available, it is very important that the selected **batterytype** value matches the batteries actually in use.

  Currently supported battery types are:

  - **lisocl2** (Li-$SOCl_2$)
  - **lifes2** (Li-FeS_2)
  - **znmno2** (Zn-MnO_2)
  - **linimnco** (Li-NiMnCo)
  - **nimh** (NiMH)
  - **none**
- **capacity** is a read-only parameter which reports the total nominal energy capacity (in Joules) of the internal battery set. It cannot be changed directly, but changes according to the selected **batterytype**.
- **used** [= 0], reports the accumulated energy used from the internal battery since the value was last reset. The value continues to be updated even if it exceeds the nominal capacity. When fresh batteries are installed the value can be reset to zero; this is the only accepted value for updating the parameter.

Examples

```
>> powerinternal
<< powerinternal batterytype = nimh, capacity = 138.000e+003, used = 100.100e+003
```

```
>> powerinternal used = 0
<< powerinternal used = 0.000e+000
```

Errors

Error E0108 invalid argument to command: '<invalid-argument>'

3.8.6 powerexternal

Usage

```
>> powerexternal [batterytype | capacity | used ]
```

Security

Open.

Description

Allows various parameters to be reported or set for the current deployment.

- **batterytype** [= <*batterytype*>], has a value corresponding to a description of the various battery types supported; see the list below. The RBR*fermata, RBRfermette* and RBR*fermette*^{3} battery packs are provided by RBR Ltd; for any other type of external power source, **other** should be used.

  The **batterytype** may be set prior to a deployment; it cannot be changed while a deployment is in progress. If proper estimates are required for battery capacity used and deployment life available, it is very important that the selected **batterytype** value matches the power source actually in use.

  Currently supported battery types are:

  - **fermata_lisocl2** (Li-$SOCl_2$-equipped RBR*fermata*)
  - **fermata_znmno2** (Zn-MnO_2-equipped RBR*fermata*)
  - **fermette_limno2** (Li-MnO_2-equipped RBR*fermette*)
  - **fermette3_lisocl2** (Li-$SOCl_2$-equipped RBR*fermette*^{3})
  - **fermette3_lifes2** (Li-FeS_2-equipped RBR*fermette*^{3})
  - **fermette3_znmno2** (Zn-MnO_2-equipped RBR*fermette*^{3})
  - **fermette3_linimnco** (Li-NiMnCo-equipped RBR*fermette*^{3})
  - **fermette3_nimh** (NiMH-equipped RBR*fermette*^{3})
  - **fermata_nimh** (NiMH-equipped RBR*fermata* - f/w version 1.140 or later)
  - **other**
- **capacity** is a read-only parameter which reports the total nominal energy capacity (in Joules) of the external battery pack. It cannot be changed directly, but changes according to the selected **batterytype**. The capacity value for the power source type **other** is currently zero, but energy used from this source will still be tracked; see below.
- **used** [= 0] reports the accumulated energy used from the external power source since the value was last reset. The value continues to be updated even if it exceeds the nominal capacity. If a fresh battery pack is installed the value can be reset to zero; this is the only accepted value for updating the parameter.

Examples

```
>> powerexternal
<< powerexternal batterytype = fermata_lisocl2, capacity = 22.000e+006, used =
100.100e+003
```

```
>> powerexternal used = 0
<< powerexternal used = 0.000e+000
```

Errors

Error E0108 invalid argument to command: '<invalid-argument>'

3.8.7 info

Usage

```
>> info [ pn ]
```

Security

Open.

Description

This is a read-only command which reports more information about the logger. Currently it reports:

- **pn**, RBR part number of the instrument
- for firmware versions 1.094 or higher, **fwlock = on|off**

The **fwlock** parameter is set at the factory, and for most instruments it is **off**, which means that the standard method of updating instrument firmware using the Ruskin software can be used. For some OEM applications requiring that the version of firmware does not change, the **fwlock** is set to on, which prevents firmware updates by the standard method.

If necessary, a special procedure can be used to override this 'locked' state; please contact RBR Ltd if you believe you need to do this.

Examples

```
>> info pn
<< info pn = 0123456revA
```

```
>> info
<< info pn = L3-M13-F15-BEC12-INT12-SCT16-SP11
```

```
>> info
<< info pn = L3-M13-F15-BEC12-INT12-SCT16-SP11, fwlock = off
>> info fwlock
<< info fwlock = off
```

For firmware versions 1.094 or higher.

Errors

None.

3.8.8 getall

Usage

`>> getall`

Security

Open.

Description

This is a read-only command which reports all the information and settings in the logger. It reports all the parameters and settings in use by the logger. It outputs the result over several lines unlike standard commands. If a feature is unavailable, its associated command result will not be part of the getall.

Examples

```
>> getall
<< serial baudrate = 115200, mode = rs232, availablebaudrates =
115200|19200|9600|4800|2400|1200|230400|460800, availablemodes =
rs232|rs485f|uart|uart_idlelow
```

```
prompt state = on
confirmation state = on
link type = usb
streamusb state = off
streamserial state = off, aux1_enabled = false, aux1_setup = 1000, aux1_hold = 1000,
aux1_active = high, aux1_sleep = tristate
power source = usb, int = 0.00, ext = 0.00
powerinternal batterytype = none, capacity = 0.000e+000, used = 0.000e+000
powerexternal batterytype = other, capacity = 0.000e+000, used = 0.000e+000
meminfo used = 834, remaining = 1056963096, size = 1056964608
memformat type = rawbin00, newtype = calbin00
settings fetchpoweroffdelay = 8000, sensorpoweralwayson = off, temperature = 15.0000,
atmosphere = 10.1325010, pressure = 10.1325, density = 1.0260209, speccondtempco =
0.0191, salinity = 35.0000, avgsoundspeed = 1506.8000, altitude = 0.0000, castdetection
= off, inputtimeout = 10000
clock datetime = 20000101041228, offsetfromutc = unknown
sampling mode = continuous, period = 20000, burstlength = 10, burstinterval = 300000,
gate = none, userperiodlimit = 500, availablefastperiods = 500
twistactivation enabled = false, state = n/a
ddsampling direction = ascending, fastperiod = 1000, slowperiod = 5000, fastthreshold =
3.0, slowthreshold = 3.0
deployment starttime = 20000101000000, endtime = 20991231235959, status = stopped
calibration 1 label = temperature_00, datetime = 20000401000000, c0 = 3.5000000e-003, c1
= -250.00002e-006, c2 = 2.7000000e-006, c3 = 23.000000e-009 || calibration 2 label =
pressure_00, datetime = 20000401000000, c0 = 0.0000000e+000, c1 = 1.0000000e+000, c2 =
0.0000000e+000, c3 = 0.0000000e+000, x0 = 0.0000000e+000, x1 = 0.0000000e+000, x2 =
0.0000000e+000, x3 = 0.0000000e+000, x4 = 0.0000000e+000, x5 = 0.0000000e+000, n0 =
value || calibration 3 label = phycoerythrin_00, datetime = 20000401000000, c0 =
0.0000000e+000, c1 = 1.0000000e+000 || calibration 4 label = seapressure_00, datetime =
20000401000000, n0 = value, n1 = value || calibration 5 label = depth_00, datetime =
20000401000000, n0 = value, n1 = value
outputformat type = caltext01, availabletypes =
caltext01|caltext02|caltext03|caltext04|caltext07, labelslist =
temperature_00|pressure_00|phycoerythrin_00|seapressure_00|depth_00
simulation state = off, period = 3600000, channels = 1|2|3
id model = RBRconcerto3, version = 1.000, serial = 999999, fwtype = 104
info pn = n/a
hwrev pcb = J, cpu = 5659A, bsl = A
channels count = 5, on = 5, settlingtime = 600, readtime = 350, minperiod = 500
channel 1 type = temp09, module = 1, status = on, settlingtime = 50, readtime = 260,
equation = tmp, userunits = C, label = temperature_00 || channel 2 type = pres24, module
= 2, status = on, settlingtime = 50, readtime = 290, equation = corr_pres2, userunits =
dbar, label = pressure_00 || channel 3 type = fluo00, module = 40, status = on,
settlingtime = 600, readtime = 350, equation = lin, userunits = ug/L, label =
phycoerythrin_00 || channel 4 type = pres08, module = 240, status = on, settlingtime =
0, readtime = 0, equation = deri_seapres, userunits = dbar, label = seapressure_00 ||
channel 5 type = dpth01, module = 241, status = on, settlingtime = 0, readtime = 0,
equation = deri_depth, userunits = m, label = depth_00
sensor 1 || sensor 2 || sensor 3 || sensor 4 || sensor 5
wifi enabled = false, state = n/a, timeout = 60, commandtimeout = 60, baudrate = 921600
postprocessing status = idle, channels =
mean(pressure_00)|mean(temperature_00)|mean(phycoerythrin_00), tstamp_min =
20000101000000, tstamp_max = 20991231235959, binsize = 1.0, binreference = tstamp,
depth_min = -10.0, depth_max= 15000.0, binfilter = none
```

Errors

None.

3.9 Data sample

3.9.1 fetch

Usage

```
>> fetch [ channels | sleepafter ]
```

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 settling time (or power-on settling delay) for the attached sensors, there may be a noticeable delay before the <*sample-data*> appears. Refer to the **channels** command for further discussion of settling time.

- **channels** [= <*listofchannels*>], is the list of channels to be acquired, using either indices or labels of channels (see **channel**). The output format of the <sample-data> is the channels samples in the same order. If this parameter is not provided, the output format of the <*sample-data*> is determined by the **outputformat** command.

> If a channel or one of its supporting channels has its status set to **off**, requesting that channel will return an Error-14 as sample data for this channel. See **channel** for details.

- **sleepafter** [= <*sleepafter*>], controls power strategy after the fetch is issued. <*sleepafter*> is either:
  - **false** (default behaviour), 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** command for information on changing the default sensor power-off timeout.
  - **true** 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**, 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** 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.

Examples

```
>> outputformat labelslist
<< outputformat labelslist = temperature_00|pressure_00
>> fetch
<< 2017-10-21 11:50:49.000, 18.1745 C, 12.7052 dbar
```

Fetch a sample from all channels.

```
>> channel 1 type
<< channel 1 type = cond07
>> channel 2 type

<< channel 2 type = temp14

>> channel 3 type

<< channel 3 type = pres24
>> outputformat type, labelslist
<< outputformat type = caltext02, labelslist =
conductivity_00|temperature_00|pressure_00

>> fetch
<< 2017-10-21 11:50:49.000, 40.0120 mS/cm, 18.1745 C, 12.7052 dbar
>> fetch channels = 3|2
<< 2017-10-21 11:50:49.000, 12.7052 dbar, 18.1745 C
>> fetch channels = pressure_00|temperature_00, sleepafter=true
<< 2017-10-21 11:50:49.000, 12.7052 dbar, 18.1745 C
```

Fetch a sample from some specific channels.

```
>> fetch
<< 2020-11-25 15:31:55.000, Error-07, ###, Error-14
```

Fetch command reporting error for channel 1 (timeout) and channel 3 (unable to compute value). Channel 2 is not calibrated. List of error code can be found in within Sample data standard format.

Errors

Error E0108 invalid argument to command: '<invalid-argument>'
An unrecognized argument was given with the command.

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

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

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

3.10 Security and Interaction

3.10.1 permit

Usage

`>> permit [ command ]`

Security

Open. Permitted commands may have further restraints (i.e., may be unsafe).

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.

It takes as a mandatory parameter:

- **command** = <commandname>, where <commandname> is the command to permit.

Examples

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

Successfully clears the data memory.

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

Fails because memclear is a protected command.

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

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

Errors

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

Error E0108 invalid argument to command: '<invalid-argument>'
The <command-name> argument given is not a recognized command.

3.10.2 prompt

Usage

```
>> prompt [ state ]
```

Security

Open.

Description

Returns the state of the "**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.

- **state** [= <state>]
  - **on**, the prompt is sent

- **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 invalid argument to command: '<invalid-argument>'
An unrecognized argument was given.

3.10.3 confirmation

Usage

```
>> confirmation [ state ]
```

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.

- **state** [= <state>]
  - **on**, the confirmation is sent
  - **off**, the confirmation is suppressed.

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:

- Requests to simply report a parameter always generate output.
- Error messages resulting from a failed attempt to set a parameter are always sent.
- Some 'action' commands such as enable always generate a confirmation message.
- The "**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.

```
>> deployment starttime
<< starttime = 20171207130000
```

All commands are with confirmation state = off. A request for information always provokes a response.

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

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

```
>> deployment starttime = 20171201120000
<<
```

Response suppressed if parameter is successfully changed.

Errors

Error E0108 invalid argument to command: '<invalid-argument>'
An unrecognized argument was given.

3.10.4 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 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 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 cannotrespond.

Examples

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

Successfully resets the logger CPU.

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

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

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

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

Errors

Error E0103 protected command, use 'permit command = <command>'
permit command = reboot must immediately precede the command to reset the logger.

3.11 Realtime data

3.11.1 streamserial

Usage

```
>> streamserial [state | aux1_state | aux1_setup | aux1_hold | aux1_active |
aux1_sleep | aux1_all ]
```

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** command. This commands 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** 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.

- **state** [= <*state*>]
  - **on**, enables the serial streaming.

    - **off**, disables the serial streaming.
- 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** 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** [= <*state*>]: 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** [= <*activelevel*>]: reports or sets the active level of the AUX1 signal seen by the external device during the setup time, data transmission and hold time, either **high** or **low**. 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** [= <*sleeplevel*>]: reports or sets the level of the AUX1 signal seen by the external device while the logger is asleep, either **high**, **low** or **tristate**. 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 invalid argument to command: '<invalid-argument>'
The supplied argument was not valid or out of place.

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

3.11.2 outputformat

Usage

```
>> outputformat [ type | availabletypes | channelslist | labelslist ]
```

Security

Unsafe.

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:

- **type** [= <*type*>] set and/or report the current output format type. Not all instruments support all the formats; query the **availabletypes** parameter to check which formats are available.
- **availabletypes** reports the formats which are available.
  At present, the formats which may be supported are:
  - **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:
    2017-09-10 11:24:14.000, 38.6664, 21.5183, 10.9601
  - **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:
2017-09-10 11:52:21.000, 38.6671 mS/cm, 22.0217 C, 10.9596 dBar

- **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:
  2017-09-10 11:52:21.000, 38.6671142, 22.0217241, 10.9596633

- **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:
  2017-09-10 11:52:21.000, 38.6671142e+000, 22.0217124e+000, 1.95962418e+003

- **caltext07**, Calibrated ASCII output. This output format is supported in firmware versions 1.109 or later.
  The output starts with the keyword RBR followed by the serial number followed by 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 16-bit CRC using the CCITT polynomial f(x) = x^16 + x^12 + x^5 + 1, feeding bytes into the generator LSB first and using 0xFFFF as a seed value, is then transmitted. All characters sent including the 'R' character of the RBR keyword and the last space character before the <crc> are used to calculate the CRC.
  RBR keyword is separated by a space from the serial number. A comma and space separate the serial number, the timestamp, values, and the CRC. The CRC is formatted in hexadecimal with a leading 0x followed by 4 digits.

Format:
RBR <sn>, YYYY-MM-DD hh:mm:ss.ttt, <value1>, <value2>, … , <crc>

Example for a 3-channel logger:
RBR 142152, 2017-09-10 11:24:14.000, 38.6664, 21.5183, 10.9601, 0xAD28

- **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	fluorometry-UV	pressure_period
BTEX	measurement_count	refined_fuels
calphase	methane	rhodamine
CDOM	optical_brighteners	salinity
chlorophyll	ORP	specific_conductivity
conductivity	PAR	speed_of_sound
crude_oil	partial_CO2_pressure	temperature
custom_fluorometer	period	temperature_period
cyanobacteria	pH	transmittance
depth	phycocyanin	turbidity
dissolved-O2	phycoerythrin	voltage
fluorescein	pressure	

- **labelslist** This reports the list of active channels labels, 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. A channel's label is set upon request for OEM customers (see **channel** command).

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'.
- **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, labelslist =
temperature_00|pressure_00|salinity_00|conductivitycelltemperature_00
```

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

```
>> outputformat availabletypes
<< outputformat availabletypes = caltext01|caltext02|caltext03|caltext04
```

```
>> outputformat channelslist
<< outputformat channelslist =
temperature(C)|pressure(dbar)|salinity(PSU)|temperature(C)
```

```
>> outputformat labelslist
<< outputformat labelslist =
temperature_00|pressure_00|salinity_00|conductivitycelltemperature_00
```

Errors

Error E0108 invalid argument to command: '<invalid-argument>'
The command was given with an argument which is unrecognized or misplaced.

3.11.3 streamusb

Usage

```
>> streamusb [ state ]
```

Security

Open.

Description

This command can turn on or off the USB streaming. When USB streaming is on, acquired data is sent over the USB link at the same time it is stored in memory. Refer to the **outputformat** command for descriptions of the data formats which may be available.

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.

This command takes a single parameter:

- **state** [= <*state*>]
  - **on**, enables the USB streaming.
  - **off**, disables the USB streaming.

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

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

Errors

Error E0108 invalid argument to command: '<invalid-argument>'

The supplied argument was not valid or out of place.

Error E0109 feature not available

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

4 Format of stored data

4.1 Overview

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

- A deployment header, which contains meta-information about the logger and the deployment parameters.
- Sample data, comprising sets of measured values from all active channels in the logger.
- 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.

Unless stated otherwise, all the numbers stored in the logger's memory are stored in little-endian format.

4.1.1 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:

- The deployment header is always stored at the beginning.
- Following the header, sample data is stored as it accumulates.
- 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** **newtype** command. When downloading data, use the **memformat** **type** command to determine the format of the data which is currently in memory. Downloading data is performed using the **readdata** command on dataset-1.

4.1.2 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 assignment of datasets to the stored information is as follows:

- The deployment header is in dataset 2.
- The sample data is in dataset 1.
- Events are in dataset 0.

The **readdata** command acting on each of these three datasets as required is used to download the information: use the **memformat 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.

4.2 EasyParse "calbin00" format

4.2.1 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:

- Most event types have no payload and the content is not defined; it should be ignored.
- A separate time-stamp field for milliseconds is redundant and never used.
- For regime bin events (0x20), the payload is the number of reading values in the reported average.
- 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
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

Hex Code	Description	Payload
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	Wi-Fi module detected and activated	undefined
0x1B	Wi-Fi module de-activated; removed or activity timeout	undefined
0x1C	Regimes enabled, but not yet in a regime	undefined
0x1D	Entered regime 1	undefined
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
0x27	Energy used marker, internal battery	energy consumed from power source since last accumulator reset

Hex Code	Description	Payload
0x28	Energy used marker, external power source	energy consumed from power source since last accumulator reset
0x29	Device control action result	result of control action

Logger configuration may also include a derived data channel, type **cnt_00**, which contains the same value as 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" for further details.

4.2.2 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:

- 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.
- 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.
- 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** 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 (e.g., 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

Error #	Hex Code	Description
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)
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
19	0xFF810013	sensor output not received within timeout
20	0xFF810014	unable to parse sensor output
21	0xFF810015	channel is not correctly calibrated
22	0xFF810016	floating point value is badly formed
23	0xFF810017	channel not logged

4.3 Standard "rawbin00" format

4.3.1 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:

- In Standard format (rawbin00), the header is stored at the beginning of dataset-1, prior to any sample data or events.
- 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
2.004	Section 3 (deployment) includes a list of simulated channels	**simulation**
2.003	Section 6 (devices) includes segmented schedule table for BPR|zero	**valve**, **valvesegments**, **valvesegment**
2.002	Added Section 6 for details of installed devices	**valve**
2.001	Added an internal parameter to Section 2	n/a
2.000	Original release of this overall format.	n/a

Version 2.004

Introduction

More recent header versions are always backward compatible with previous versions, in the sense that while new material may be added, existing content and the layout of existing content is not changed. The following table lists all versions of the header, the version of firmware in which it was first introduced, and a brief description of the change.

Header version	Firmware version	Comment
2.004	1.150	**Deployment section** includes a list of simulated channels
2.003	1.142	**Device section** includes segmented schedule table for BPR|zero
2.002	1.115*	Added **Device section (Section Id 0x06)** for BPR|zero and UV-LED
2.001	1.080	Added an internal parameter to Section 2
2.000	1.000	Original release

*Prior to Version 1.130, only specific versions of the firmware include support for the BPR|zero product; there will be some versions of firmware that do not include a Section 6 at all, so do not assume that it will be present but empty.

Overview

The header is composed of a series of sections followed by a global CRC. Each section describes some aspect of the instrument configuration or deployment settings. Sections should be in ascending order by Section Id value. The 'metadata' section describes the data format itself.

Header		
Field	**Length**	**Notes**
Metadata section	9 bytes	The metadata section is always the first section in the header, and currently has a fixed size
Other sections	variable	
CRC	2 bytes	16-bit CRC using the CCITT polynomial f(x) = x^16 + x^12 + x^5 + 1. Header bytes are fed into the LSB of the generator.

Each section is organized as:

Section structure		
Field	**Length**	**Notes**
Section Id	1 byte	Section Id identifies the kind of information stored in this section 1. 0x01 Metadata section 2. 0x02 Logger section 3. 0x03 Deployment section 4. 0x04 Non deployment related settings section 5. 0x05 Channels section 6. 0x06 Devices section
Length	2 bytes	This is the total length of the section (including **Section Id**, **Length** and **Content** fields)
Content	(**Length** - 3) bytes	The content of each section is described below. If there are any unused bytes, they are set to 0xFF.

Metadata section content (Section Id 0x01)

Version (4 bytes)

For example, 2002; this is the version of the header format

Total header length (2 bytes)

This length includes all the sections of the header and the CRC at the end.

Logger section content (Section Id 0x02)

Firmware type

The firmware type code as an 32b integer; see the fwtype parameter under the id command.

Firmware version

Given as an integer; for example version 1.570 would be given as 1570.

Logger Serial Number

As reported by the **id** command.

Logger model (16 bytes)

NUL terminated string reflecting the model name. Padded with 0xFF bytes.

Logger P/N length (2 bytes)

Size in bytes of the logger P/N string (including the NUL character).

Logger P/N string (**Logger P/N length** bytes)

NUL terminated string reflecting the P/N.

Power supply P/N length (2 bytes)

Size in bytes of the power supply P/N string (including the NUL character).

Power supply P/N string (**Power supply P/N length** bytes)

NUL terminated string reflecting the power supply P/N.

Reserved (8 bytes)

Deployment section content (Section Id 0x03)

Dataset format

0x00000000 rawbin00
0x00000001 calbin00

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, (946684800 Unix timestamp).

Schedule start time

Programmed by the **deployment** **starttime** command. The value is the number of seconds elapsed since 2000-Jan-01 00:00:00, (946684800 Unix timestamp).

Schedule end time

Programmed by the **deployment** **endtime** command. The value is the number of seconds elapsed since 2000-Jan-01 00:00:00, (946684800 Unix timestamp).

Measurement interval

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

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 **deployment status** command.

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.
0x00000200 Reserved
0x00000400 1 = firmware updates are locked
0x00000800 Reserved
0x00001000 1 = valve schedule started with no delay (if supported by firmware)
0x00002000 1 = valve schedule enabled (if supported by firmware)
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.
0x00200000 1 = the directional dependent sampling mode was enabled.
0x00400000 1 = the Wi-Fi link was enabled.

Burst/average interval

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

Burst/average length

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

Thresholding channel index

The channel monitored for threshold-gated sampling, if enabled (see Feature flags above).

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.

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.

Reference pressure for Wi-Fi 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.

Offset from Universal Coordinated Time (UTC)

This parameter 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 **now** command to update the date/time, it will be reported as a NaN ('Not a Number' in IEEE floating point format).

Simulated data cycle period

When a logger is enabled in simulation mode, the simulated data is periodic, with the period usually 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 .

Directional dependent sampling flags

This is a bitmask of settings for directional dependent sampling.
0x00000001 1 = ascending direction, 0 = descending direction.

Directional dependent fast sampling period

The fast sampling period in milliseconds.

Directional dependent slow sampling period

The slow sampling period in milliseconds.

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.

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.

Battery type, internal

An integer encoding the type of the internal battery as set by the **powerinternal** **batterytype** command, and as represented in the following table, which also shows the nominal **Battery capacity** for each battery type.

Battery type code	Type name
0	**none**
1	(reserved)
2	**lisocl2**
3	**lifes2**
4	**znmno2**
5	**linimnco**

Battery type code	Type name
6	**nimh**

Battery type, external

An integer encoding the type of the internal battery as set by the **powerexternal batterytype** command, and as represented in the following table, which also shows the nominal **Battery capacity** for each battery type.

Battery type code	Type name
100	**other**
101	**fermata_lisocl2**
102	**fermata_znmno2**
103	**fermette_limno2**
104	**fermette3_lisocl2**
105	**fermette3_lifes2**
106	**fermette3_znmno2**
107	**fermette3_linimnco**
108	**fermette3_nimh**

Battery capacity, internal

This is an IEEE 32-bit (single precision) floating point number representing the nominal total energy capacity in Joules of the indicated internal **Battery type**; see above.

Battery capacity, external

This is an IEEE 32-bit (single precision) floating point number representing the nominal total energy capacity in Joules of the indicated external **Battery type**; see above.

Energy used from internal battery

This is an IEEE 32-bit (single precision) floating point number representing an estimate of the total energy in Joules removed from the internal battery since the value was last reset (zeroed).

Energy used from external battery

This is an IEEE 32-bit (single precision) floating point number representing an estimate of the total energy in Joules used from the external power source since the value was last reset (zeroed).

Energy consumption parameter e1

This is an IEEE 32-bit (single precision) floating point number representing an estimate of the energy in Joules required to take a single power-cycled sample.

Energy consumption parameter p1

This is an IEEE 32-bit (single precision) floating point number representing an estimate of the power in Watts used by the logger in sleep mode.

Energy consumption parameter p2

This is an IEEE 32-bit (single precision) floating point number representing an estimate of the power in Watts used by the logger when sampling while continuously powered on.

Energy consumption parameter p3

This is an IEEE 32-bit (single precision) floating point number representing an estimate of the power in Watts used by the logger while using an alternative communications link such as Wi-Fi.

[*Other energy consumption parameters may be present; see below. They are not grouped together in order to keep the header layout compatible between different firmware versions.*]

Specific Conductivity Temperature Coefficient

This parameter 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.

Temperature default value

This item is a floating point number in IEEE 32-bit (single precision) format, not an integer. The units must be Celsius.

Pressure default value

This item is a floating point number in IEEE 32-bit (single precision) format, not an integer. The units must be dbar.

Atmospheric pressure default value

This item is a floating point number in IEEE 32-bit (single precision) format, not an integer. The units must be dbar.

Density default value

This item is a floating point number in IEEE 32-bit (single precision) format, not an integer. The units must be kg/m^3.

Salinity default value

This item is a floating point number in IEEE 32-bit (single precision) format, not an integer. The units must be PSU.

Average sound speed default value

This item is a floating point number in IEEE 32-bit (single precision) format, not an integer. The units must be m/s.

Altitude default value

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 **settings** command.

Energy consumption parameter p4

This is an IEEE 32-bit (single precision) floating point number representing an estimate of the power in Watts used by the logger while an installed 'device' is active. Current examples of 'devices' include the UV-antifouling feature and the valve controller for the BPR|zero system.

Simulated channels mask

This is a 32-bit mask with one bit per channel indicating whether the channel was simulated (bit is 1) or has true measured data (bit is 0). Only measured channels can be simulated, but the simulated data will be used by any derived channels dependent on the simulated channels.
Channel 1 is represented by bit 0 (the least significant bit), Channel 2 by bit 1, and so on.

Non deployment related settings section content (Section Id 0x04)

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

Serial link baudrate

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

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.

Power-on timeout for Wi-Fi module, if installed

After activating the Wi-Fi 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 Wi-Fi module will be deactivated again.

Command timeout for Wi-Fi 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 Wi-Fi module will be deactivated.

Fetch power off delay

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

Channel section content (Section Id 0x05)

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

Label (32 bytes)

This is a character ASCII string NUL terminated. If label is not set, all bytes are set to 0xFF.

Module fwtype (32 bytes)

This is a character ASCII string NUL terminated. If the fwtype is unknown, all bytes are set to 0xFF.

Firmware version

Given as an integer; for example version 1.570 would be given as 1570.

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 cannot 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.
Coefficients appear in ascending order in the following manner: C-coefficients first, then X-coefficients, then N-coefficients.
Refer to the **calibration** command for more details.

Channel specific structures total size (2 bytes)

Applies to channels with gain settings, **sensor** key/value pairs, or other channel-specific information. This is the total size of all the channel-specific structures.

Channel specific structures

Applies to channels with gain settings, **sensor** key/value pairs and specific frontends.

This is a list of specific structures which applies to a channel. Each structure starts with a header:

Channel specific structure		
Field	**Length**	**Notes**

Channel specific structure		
Type	(1 byte)	an integer indicating the type of information stored.
Length	(2 bytes)	This is the total length of the channel specific structure (including **Type**, **Length**, **Spare,** and **Content** fields.
Spare	(2 bytes)	Always 0.
Content	(**Length**-5)	The content of this field is specified below for each type.

Channel specific structure Type 2 Content- Key/value pair attached to sensor command		
Field	**Length**	**Notes**
Sensor Key	(variable)	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.
Sensor Value	(variable)	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.

Channel specific structure Type 3 Content - Channel gain control		
Field	**Length**	**Notes**
Ranging mode	(1 byte)	This is the status of the gain control on the sensor card. It can have only one of two values: 1 = **manual** , 2 = **auto** . See also the **channel <index> gain** command.

Channel specific structure Type 3 Content - Channel gain control		
Number of available gains	(1 byte)	This is the number of possible gains available.
Current Gain value	(4 bytes)	This is the gain currently in use at time of deployment. If the ranging mode is **auto** then this value is the first gain in the Available gain 1 . If the ranging mode is **manual** 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. This is an IEEE 32-bit (single precision) floating point number.
…		
Available gain n	(4 bytes)	This is one possible gain value available to the sensor. This is an IEEE 32-bit (single precision) floating point number.

Device section content (Section Id 0x06)

This section will not appear in header versions earlier than 2.002. In future, for versions 2.002 or later, if there are no devices configured in the instrument this section may either be a) not present at all, or b) present, but indicate that zero devices are installed, in which case there will be no supporting data to follow.

Number of devices

(1 byte) Number of devices configured in the instrument.

Device 1 offset

(2 bytes) Offset of the details for Device 1, from the beginning of the device section.

Device 2 offset

(2 bytes) Offset of the details for Device 2, from the beginning of the device section.

…

Device N offset

(2 bytes) Offset of the details for Device N, from the beginning of the device section.

[Details for device 1]

(Variable size) Depends on the type of device installed.

[Details for device 2]

(Variable size) Depends on the type of device installed.

…

[Details for device N]

(Variable size) Depends on the type of device installed.

Padding

Following the **Details for device N** section there will be from 0 to 3 bytes of padding to bring the total size of the entire deployment header up to a multiple of four bytes, if necessary. These bytes are included in the size byte-count for this section, as well as the size byte-count for the complete deployment header, but they serve no other purpose and their value is not defined. Currently, this padding appears only in the Device section, Section 6; deployment headers which do not have this section are not padded in the same way, and in general their size will not be a multiple of four bytes.

Structure of details for each device

Type

(1 byte) This is a byte code which identifies the type of device installed. An unknown or unidentified device has the code 255 (0xFF).

Module address

(1 byte) This is the internal FE-bus address of the module that interfaces to the device itself.

…….Reserved

……..(2 bytes)

Label

(24 bytes) This is a NUL-terminated ASCII character string. If no label has been set, all bytes will be 0xFF.

Module firmware type

(32 bytes) This is a NUL-terminated ASCII character string. If the firmware type is unknown, all bytes are set to 0xFF.

…… Module firmware version

…….(4 bytes) Given as a 32b integer; for example version 1.570 would be given as 1570.

Control and status flags

(2 bytes) The following flags are currently defined:

Flag	Function
Bit 0	1 = User commands to the device are enabled (always true)
Bit 1	1 = Scheduled operation of the device is enabled

Flag	Function
Bit 2	1 = Scheduled operation starts with no delay
Bit 3	1 = Store device-related events in memory during deployment
Bit 4	1 = Device schedule is segmented (f/w version 1.142 or later)
Bit 5	1 = **interval**, **duration** parameters stored internally in seconds (f/w version 1.142 or later)

Interval between scheduled device episodes

(4 bytes) An unsigned 32b integer giving the time between scheduled device episodes in milliseconds / seconds (see Bit 5 flag above).

Duration of scheduled device episodes

(4 bytes) An unsigned 32b integer giving the duration of each scheduled device episode in milliseconds / seconds (see Bit 5 flag above).

Power on delay

(4 bytes) An unsigned 32b integer giving the delay in milliseconds between powering up the external device and sending it any commands.

Power off guard time

(4 bytes) An unsigned 32b integer giving the delay in milliseconds between sending the last command to the external device and removing its power.

Size of device

(2 bytes) Indicates how many bytes of additional information follow, if any; these are specific to this particular type of device.

Device-specific details

(N bytes) Additional information specific to this type of device; the value of 'N' is indicated by the preceding size parameter.

4.3.2 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), 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 cannot 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.

Error #	Hex Code	Description
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

Error #	Hex Code	Description
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
19	0xF613F6C4	sensor output not received within timeout
20	0xF614615D	unable to parse sensor output
21	0xF615506E	channel is not correctly calibrated
22	0xF616033B	floating point value is badly formed
23	0xF6173208	channel not logged

4.3.3 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 8 … 9	16-bit 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 (for *N* > 3)

Seconds for the date/time are counted from 2000-01-01T00:00:00Z.

Event processing info byte

Bits	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
Bits 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
0x05	One or more parameters recovered after reset
0x06	Restart failed: real-time clock (RTC)/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 RTC
0x0B	Parameters recovered; sampling restarted after resetting RTC
0x0C	Sampling finished: deployment end time reached
0x0D	Start of a recorded burst
0x0E	Start of a wave burst
0x0F	Power source switched to USB
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

Type code	Description
0x17	Power source switched to external battery
0x18	Twist activation started sampling
0x19	Twist activation paused sampling
0x1A	Wi-Fi module detected and activated
0x1B	Wi-Fi 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
0x27	Energy used marker, internal battery
0x28	Energy used marker, external power source
0x29	Device control action result
0x2A	Paused deployment resumed by the **resume** command
0x2B	Deployment paused using the **pause** command

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.

0x20 Regime bin event auxiliary data

Bytes	Description
Bytes 12 … 15	Number of readings in the bin

There is also 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 " for further details.

0x21 Begin profiling "up" cast event auxiliary data

Bytes	Description
Bytes 12 … 15	32-bit (4-byte) address of the corresponding sample set

0x22 Begin profiling "down" cast event auxiliary data

Bytes	Description
Bytes 12 … 15	32-bit (4-byte) address of the corresponding sample set

0x23 End of profiling cast event auxiliary data

Bytes	Description
Bytes 12 … 15	32-bit (4-byte) address of the first sample set after the cast

0x03 Runtime error event auxiliary data

Bytes	Description
Bytes 12 … 15	32-bit (4-byte) firmware program address at which the error was detected

0x27 Energy used marker, internal battery

0x28 Energy used marker, external power source

Bytes	Description
Bytes 12 … 15	32-bit (4-byte) IEEE float, energy consumed from power source since last accumulator reset.

0x29 Device control action result

Bytes	Description
Byte 12	Primary data
Byte 13	Secondary data
Byte 14	RBR proprietary device type code
Byte 15	Undefined.

The primary data byte is defined by:

Byte value	Description	Example
0x01	Device episode startaction sent	Move valve to Atmospheric position
0x02	Device episode endaction sent	Return valve to Marine position
0x20	Manual activate-device command sent	Turn UV-LEDs on
0x21	Manual deactivate-device command sent	Turn UV-LEDs off
0xFD	Start of schedule segment in segmented mode	

The secondary data byte is defined by:

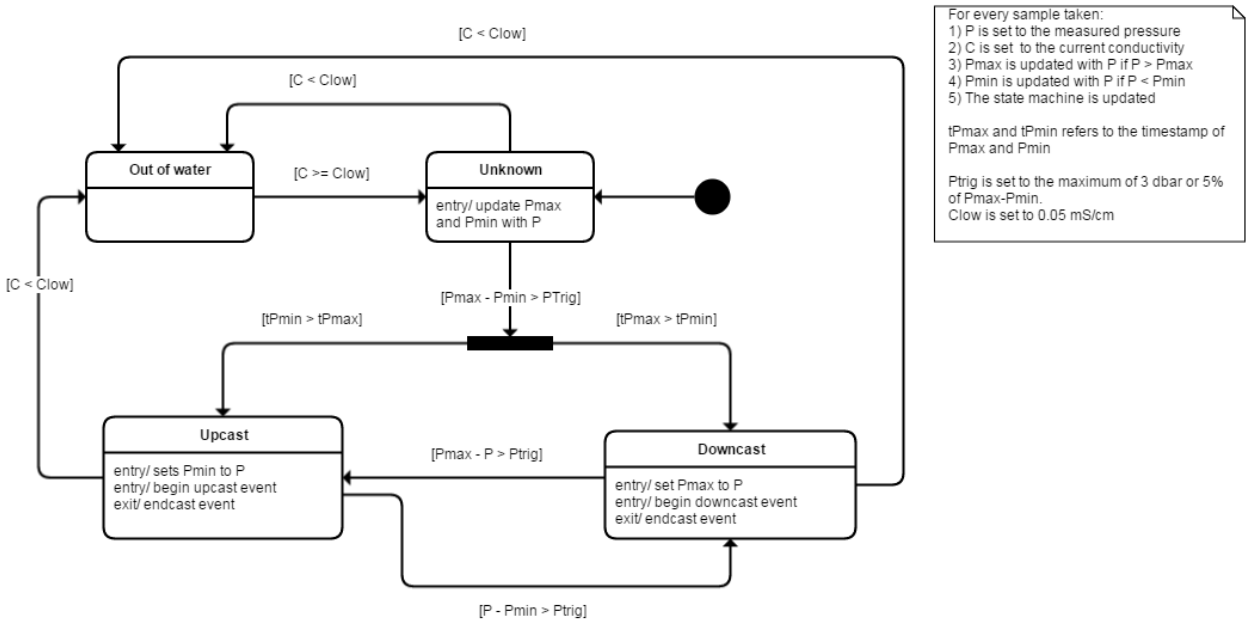
Byte value	Description
0x03	Command result - Success
0x04	Command result - Failed
0x05	Command result - Timeout
0x06	Command result - Error
0x80	Command result - Device-specific result
0x81	Command result - Device-specific result
0x01 … 0x04	Index of segment started in segmented schedule mode

4.4 Profile detection events generation

When the logger is configured with **settings castdetection = on**, it generates cast events in the recorded data (see EasyParse format events markers and Standard format events markers).

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.



5 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** command.

Type	Equation	Units	Manufacturer	Description
acc_00	linear	g	RBR	Acceleration (in g's)
alti00	distancefromechotiming	m	RBR	Distance from echo timing
baro00	linear double	ps	RBR	Frequency counter[2]
baro01	linear double	ps	RBR	Frequency counter[2]
baro02	deri_bprpres	dbar	Paroscientific/RBR	BPR calculated pressure[2]
baro03	deri_bprtemp	°C	Paroscientific/RBR	BPR calculated temperature[2]
bpr_08	deri_bprpres	dbar	Paroscientific/RBR	BPR calculated pressure[2]
bpr_09	deri_bprtemp	°C	Paroscientific/RBR	BPR calculated temperature[2]
cnt_00	none	counts	RBR	Measurement count, useful for e.g., regimes bins
cond08	corr_cond3	µS/cm	RBR	Fresh water conductivity
cond09	corr_cond	mS/cm	RBR	Marine conductivity
cond10	corr_cond	mS/cm	RBR	Marine conductivity, Combined C.T cell
cond11	corr_cond1	mS/cm	RBR	Marine conductivity, Deep combined C.T cell
cond12	corr_cond2	mS/cm	RBR	Marine conductivity, 2000dbar Combined C.T cell
cond13	corr_cond3	mS/cm	RBR	Marine conductivity, RBR*legato* C.T cell
cond14	corr_cond3	mS/cm	RBR	Marine conductivity, RBR*saildrone* C cell
cond15	corr_cond3	mS/cm	RBR	Marine conductivity, UV Antifouling CTD
cond16	corr_cond3	mS/cm	RBR	Marine conductivity, 6000dbar C cell
cond17	corr_cond3	mS/cm	RBR	Marine conductivity, 6000dbar Combined C.T cell
cond18	corr_cond3	mS/cm	RBR	Marine conductivity

Type	Equation	Units	Manufacturer	Description
cond19	corr_cond3	mS/cm	RBR	Marine conductivity, Combined C.T cell
cond21	corr_cond3	mS/cm	RBR	Marine conductivity, 2000dbar Combined C.T cell
cond22	corr_cond3	mS/cm	RBR	Marine conductivity, RBR*legato* C.T cell
cond24	corr_cond3	mS/cm	RBR	Corrected Marine conductivity (inductive),transverse C.T cell, Ti
cond25	corr_cond3	mS/cm	RBR	Corrected Marine conductivity (inductive), transverse C.T cell, POM
cond26	corr_cond3	mS/cm	RBR	Corrected Marine conductivity (inductive), transverse C.T cell, OSP
cond28	corr_cond4	mS/cm	RBR	Marine conductivity, Deep combined C.T cell, improved correction for > 2000dbar.
doxy03	linear	%	Oxyguard	Dissolved oxygen saturation
doxy07	linear	%	Aanderaa	Optode: dissolved oxygen saturation, serial
doxy08	corr_rinkoB	%	JFE Advantech	Rinko-III (B): dissolved oxygen saturation with temperature
doxy09	corr_rinkoBT	%	JFE Advantech/RBR	Rinko-III (B): dissolved oxygen saturation without temperature
doxy10	linear	µmol/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	µmol/L	RBR	Dissolved oxygen concentration compensated, serial RBR*coda* ODO
doxy25	linear	%	Oxyguard	Dissolved oxygen (saturation)
doxy27	corr_o2conc_garcia	µmol/L	RBR	Dissolved oxygen concentration compensated, serial RBR*coda* ODO|slow

Type	Equation	Units	Manufacturer	Description
doxy28	corr_o2conc_garcia	µmol/L	RBR	Dissolved oxygen concentration compensated, serial RBR*coda* ODO|fast
doxy32	corr_rinkoB2	%	JFE Advantech	Rinko-III (B): dissolved oxygen saturation with temperature
doxy33	corr_o2conc_garcia	µmol/L	RBR	Dissolved oxygen concentration compensated, serial RBR*coda* ODO, angled
doxy36	corr_o2conc_garcia	µmol/L	RBR	Dissolved oxygen concentration compensated, serial RBR*coda* ODO|low-noise
dpth01	deri_depth	m	-	Derived depth
echo01	lin	ms	RBR	Echo timing
eco_00	lin	counts	WET Labs	Generic channel for WET Labs ECO Triplet/Puck sensors
fluo00	linear	µg/L	Seapoint	Fluorometry-Phycoerythrin
fluo01	linear	µg/L	Seapoint	Fluorometry-Chlorophyll
fluo02	linear	µg/L	Seapoint	Fluorometry-Rhodamine
fluo03	linear	µg/L	Seapoint	Fluorometry-UV/CDOM
fluo04	linear	µg/L	Seapoint	Fluorometry-Phycocyanin
fluo10	linear	µg/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

Type	Equation	Units	Manufacturer	Description
fluo19	linear	cells/mL	Turner Designs	Fluorometry-Phycocyanin
fluo20	linear	cells/mL	Turner Designs	Fluorometry-Phycoerythrin
fluo21	linear	V	Turner Designs	Fluorometry-custom
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)
fluo35	linear	counts	WET Labs	Phycoerythrin (ECO Puck)
fluo36	linear	counts	WET Labs	Rhodamine (ECO Puck)
fluo37	linear	counts	Turner Designs	Fluorescence, minimum (Fo)
fluo38	linear	counts	Turner Designs	Fluorescence, maximum (Fm)
fluo39	linear	counts	Turner Designs	Fluorescence, variable (Fv)
fluo40	linear	percent	Turner Designs	Fluorescence, yield
fluo41	linear	µg/L	Seapoint	Fluorometry-Fluorescein
fluo42	linear	counts	WET Labs	Chlorophyll a (ECO FLNTU)
fluo43	linear	µg/L	RBR	Tridente Chlorophyll a

Type	Equation	Units	Manufacturer	Description
fluo44	linear	ppb	RBR	Tridente fDOM
fluo47	linear	counts	WET Labs	Phycocyanin (ECO Puck)
fluo55	linear	µg/L	RBR	Tridente Rhodamine
fluo56	linear	µg/L	RBR	Tridente Crude oil
fluo57	linear	µg/L	RBR	Tridente Phycocyanin
fluo58	linear	µg/L	RBR	Tridente Fluorescein
fluo59	linear	µg/L	RBR	Tridente Phycoerythrin
hdng00	linear	degrees	RBR	Euler angle - heading
irr_00	optic2	µW/cm²/nm	Satlantic	Irradiance
irr_01	corr_irr	V	RBR	Irradiance, 490nm
irr_02	corr_irr	µW/cm²/nm	RBR	Irradiance, 475nm
irr_05	corr_irr2	µW/cm²/nm	RBR	Irradiance (analog sensor), generic
irr_06	linear	µW/cm²/nm	RBR	Irradiance (serial sensor), generic
mag_00	linear	µT	RBR	Magnetic field strength (in micro-Tesla's)
meth00	corr_metsmet	µmol/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
opt_15	linear	°	RBR	Calibrated phase output from RBR*coda* ODO|fast
opt_24	linear	°	RBR	Calibrated phase output from RBR*coda* ODO, angled

Type	Equation	Units	Manufacturer	Description
opt_35	linear	°	RBR	Calibrated phase output from RBR*coda* ODO, low-noise
orp_01	linear	V	Idronaut	ORP (Oxidation/Reduction Potential)
orp_02	linear	V	Idronaut	ORP (Oxidation/Reduction Potential) - cabled version
orp_03	linear	V	RBR	ORP (Oxidation/Reduction Potential) - bulkhead version
orp_04	linear	V	RBR	ORP (Oxidation/Reduction Potential) - cabled version
orp_05	linear	V	AMT	ORP (Oxidation/Reduction Potential) - cabled version
par_00	linear	µmol/m²/s	Licor	PAR (photosynthetically active radiation)
par_01	linear	µmol/m²/s	Biospherical	PAR (photosynthetically active radiation)
par_03	optic2	µmol/m²/s	Satlantic	PAR (photosynthetically active radiation)
par_05	corr_irr2	µmol/m²/s	RBR	PAR (photosynthetically active radiation), analog sensor
par_06	linear	µmol/m²/s	RBR	PAR (photosynthetically active radiation), serial sensor
pco200	linear	ppm	Turner Designs	C-sense CO2 partial pressure
peri00	linear double	ps	RBR	Frequency counter[2]
peri01	linear double	ps	RBR	Frequency counter[2]
ph__00	linear	pH_units	AMT	pH
ph__01	linear	pH_units	Idronaut	pH
ph__02	corr_ph	pH_units	Idronaut	Corrected pH
ph__04	corr_ph	pH_units	Idronaut	Corrected pH, cabled version
ph__05	corr_ph	pH_units	RBR	Corrected pH, bulkhead version
ph__06	corr_ph	pH_units	RBR	Corrected pH, cabled version
ph__07	corr_ph	pH_units	AMT	Corrected pH, cabled version

Type	Equation	Units	Manufacturer	Description
phas00	linear	degrees	Aanderaa	Calibrated phase output from serial AADI Optode
pres08	deri_seapres	dbar	-	Derived hydrostatic (sea) pressure
pres23	linear	dbar	RBR	Pressure (absolute, temperature compensated), serial RBR*solo* D|rt, RBR*duet* T.D|rt
pres24	corr_pres2	dbar	RBR	Pressure (absolute, temperature compensated)
pres28	corr_pres5	dbar	RBR	Pressure (absolute, temperature compensated)
pres29	corr_pres2	dbar	RBR	Pressure (absolute, temperature compensated), UV Antifouling CTD
pres30	linear	dbar	Keller	PA10 Pressure sensor
ptch00	linear	degrees	RBR	Euler angle - pitch
roll00	linear	degrees	RBR	Euler angle - roll
sal_00	deri_salinity	PSU	-	Derived salinity, PSS78.
sal_01	deri_dyncorrS	PSU	-	Derived salinity, PSS78 with applied dynamic corrections for RBR*argo*^{3} C.T.D
scon00	deri_speccond	µS/cm	-	Derived specific conductivity
sos_00	deri_sos	m/s	-	Derived speed of sound, UNESCO, Chen and Millero
temp01	corr_rinkotemp	°C	JFE Advantech	Temperature, Rinko-III
temp04	temp	°C	RBR	Temperature (T-string)
temp06	corr_metstemp	°C	Franatech	Temperature, METS
temp07	linear	°C	Aanderaa	Temperature, from serial Optode
temp09	temp	°C	RBR	Temperature
temp13	linear	°C	RBR	Temperature, serial RBR*solo* D|rt, RBR*duet* T.D|rt
temp14	temp	°C	RBR	Temperature, combined C.T cell

Type	Equation	Units	Manufacturer	Description
temp16	linear	°C	RBR	Temperature, serial RBR*coda* ODO
temp17	linear	°C	RBR	Temperature, serial RBR*coda* ODO|slow
temp19	temp	°C	RBR	Temperature (fast)
temp22	temp	°C	RBR	Temperature, compensating channel for conductivity
temp23	linear	°C	Campbell	Temperature, OBS-501
temp26	temp	°C	RBR	Temperature RBRlegato (1s)
temp27	temp	°C	RBR	Temperature RBRlegato (fast response)
temp30	temp	°C	RBR	Temperature, compensating channel for irradiance (RBR 490nm Radiometer)
temp31	temp	°C	RBR	Temperature, standard, UV Antifouling CTD
temp32	temp	°C	RBR	Temperature, standard, 6000dbar Combined C.T cell
temp33	temp	°C	RBR	Temperature, fast, 6000dbar Combined C.T cell
temp34	temp	°C	RBR	Temperature, compensating channel for Marine C and CT 6000m
temp35	temp	°C	RBR	Temperature, fast, transverse CTD
temp36	temp	°C	RBR	Temperature, standard, transverse CTD
temp37	linear	°C	RBR	Temperature, serial RBR*coda* ODO, angled
temp38	deri_dyncorrT	°C	RBR	Temperature with C-T lag correction applied for RBR*argo*^{3} C.T.D
temp40	linear	°C	RBR	Tridente temperature
temp42	linear	°C	RBR	Tu-25 temperature
temp44	linear	°C	Keller	PA10 Pressure sensor temperature
temp51	linear	°C	RBR	Temperature, serial RBR*coda* ODO|low-noise

Type	Equation	Units	Manufacturer	Description
tran00	linear	trans_units	generic	Transmittance[1]
tran01	linear	%	WET Labs	Transmittance
tran02	linear	V	WET Labs	Transmittance
tran03	linear	1/m	Sequoia	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
turb06	cubic	FTU	Seapoint	Turbidity, STM-S S-channel
turb07	cubic	NTU	Seapoint	Turbidity, STM-S R-channel
turb08	linear	NTU	Campbell	Turbidity, OBS-501 optical backscatter
turb09	linear	NTU	Campbell	Turbidity, OBS-501 suspended solids
turb10	linear	counts	WET Labs	Turbidity (ECO FLNTU)
turb11	linear	V	Sequoia Scientific	Turbidity, LISST ABS acoustic backscatter
turb12	linear	FTU	RBR	Tridente turbidity
turb13	linear	FTU	RBR	Tridente turbidity
turb14	linear	1/m	RBR	Tridente backscatter
turb19	linear	FTU	RBR	Tu-25 turbidity, S-signal
turb20	linear	FTU	RBR	Tu-25 turbidity, R-signal
turb24	linear	1/m/sr	RBR	Tridente backscatter
turb30	linear	FTU	RBR	Tridente turbidity, S-signal, 650nm
volt00	linear	V	generic	Voltage (0V to +5V)
volt01	linear	V	generic	Voltage (–2.5V to +2.5V)
volt02	linear	V	generic	Voltage (0V to +10V)

Type	Equation	Units	Manufacturer	Description
volt03	linear	V	generic	Voltage (–10V to+10V)

[1]Includes SeaTech and WET Labs.

[2]**bpr_08**, **bpr_09**, **peri00**, and **peri01** are calculated using double precision in order to maintain resolution.

6 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.

6.1 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.

6.1.1 lin, or linear

$$output = c_0 + c_1 \cdot R$$

6.1.2 qad, or quadratic

$$output = c_0 + c_1 \cdot R + c_2 \cdot R^2$$

6.1.3 cub, or cubic

$$output = c_0 + c_1 \cdot R + c_2 \cdot R^2 + c_3 \cdot R^3$$

6.1.4 tmp, or temperature

Given in °C, based on the Steinhart-Hart equation used for thermistors.

$$T = \frac{1}{Y} - 273.15$$

where

$$Y = c_0 + c_1 \cdot X + c_2 \cdot X^2 + c_3 \cdot X^3$$

and

$$X = ln(\frac{1}{R} - 1)$$

6.2 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.

6.2.1 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); 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:

$$T = c_0 + c_1 \cdot V + c_2 \cdot V^2 + c_3 \cdot V^3$$

where

$$V = x_0 + x_1 \cdot R$$

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.

```
>> calibration 5 type
<< calibration 5 type = temp01
```

Confirm the channel type.

```
>> calibration 5 datetime = 20171201120000, c0 = -5.65608, c1 = 16.80047, c2 =
-2.253705, c3 = 0.4827284
```

Set the core coefficients for the temperature output.

```
>> calibration 5 datetime = 20171201120005, x0 = 6.782656, x1 = -9.257345
```

Set the secondary coefficients for the voltage conversion.

```
>> calibration 5
<< calibration 5 type = temp01, datetime = 20171201120005, c0 = -5.65608, c1 = 16.80047,
c2 = -2.253705, c3 = 0.4827284, x0 = 6.782656, x1 = -9.257345
```

Request confirmation of all calibration coefficients.

6.2.2 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); 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:

$$T = c_0 + c_1 \cdot V$$

where

$$V = x_0 + x_1 \cdot R$$

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.

```
>> calibration 5 type
<< calibration 5 type = temp06
```

Confirm the channel type.

```
>> calibration 5 datetime = 20171201000000, c0 = -5.65608, c1 = 16.80047
```

Set the core coefficients for the temperature output.

```
>> calibration 5 datetime = 20171201000000, x0 = 6.782656, x1 = -9.257345
```

Set the secondary coefficients for the voltage conversion.

```
>> calibration 5
<< calibration 5 type = temp06, datetime = 20171201000000, c0 = -5.65608, c1 = 16.80047,
x0 = 6.782656, x1 = -9.257345
```

Request confirmation of all calibration coefficients.

6.2.3 optic2 - optical parameters measured by a Satlantic OCR sensor

RBR data loggers support the integration of the Satlantic OCR-504 sensor for measurement of optical parameters such as PAR, irradiance, and radiance.

This is not 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, as a convenient means of implementing an internal scale factor. The **c**-group coefficients correspond directly to sensor coefficients provided by the manufacturer; the **x0** coefficient is used only as an internal scaling factor, and its value must not be changed.

The equation is:

$$output = c_2 \cdot c_1 \cdot ((R \cdot x_0) - c_0)$$

where

- **R** is the logger's reported voltage ratio,
- **x0** has the fixed value 2^32 (4294967296) and must not be changed.
- **c0** corresponds exactly to the Satlantic coefficient `a0`,
- **c1** corresponds exactly to the Satlantic coefficient `a1`
- **c2** corresponds exactly to the Satlantic coefficient `Im`.

```
>> calibration 5 type
<< calibration 5 type = irr_00
```

Confirm the channel type.

```
>> calibration 5 datetime = 20171201000000, c0 = 2147559862.6, c1 = 1.52173169935e-7, c2
= 1.161
<< calibration 5 datetime = 20171201000000, c0 = 2.1475600e+009, c1 = 152.17317e-009, c2
= 1.1610001e+000
```

Set the core coefficients for the sensor output.

```
>> calibration 5
<< calibration 5 type = irr_00, datetime = 20171201000000, c0 = 2.1475600e+009, c1 =
152.17317e-009, c2 = 1.1610001e+000, x0 = 4.2949670e+009
```

Request confirmation of all calibration coefficients.

6.3 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 cannot 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 that 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 examples 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.

6.3.1 Example 1: corr_pH - simple temperature correction of pH

Consider an RBR*concerto*^{3} C.T.D.pH logger. Without temperature correction, the pH output from Channel-4 would be a simple linear function of the raw data:

$$pHraw = c_0 + c_1 \cdot R$$

where R is the normalized voltage ratio from Channel-4 monitoring the pH sensor, **c0**, **c1** are the core coefficients of the linear equation, and **pHraw** 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:

$$pHcorr = pHraw + Kph \cdot (pHraw - pHcal) \cdot (T - Tcal)$$

Casting this into the form used by the logger, corr_ph, is simple:

$$pHcorr = pHraw + x_0 \cdot (pHraw - x_1) \cdot (value(n_0) - x_2)$$

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.

Examples

```
>> calibration 4 type
<< calibration 4 type = ph__02
```

Confirm the channel type.

```
>> calibration 4 datetime = 20171201000000, c0 = 15.23461, c1 = -0.198743
```

Set the core coefficients.

```
>> calibration 4 datetime = 20171201000000, x0 = -0.00302, x1 = 7, x2 = 24.943
```

Set the cross-channel correction coefficients.

```
>> calibration 4
<< calibration 4 type = ph__02, datetime = 20171201000000, c0 = 15.23461, c1 =
-0.198743, x0 = -0.00302, x1 = 7, x2 = 24.943, n0 = 2
```

Request confirmation of all calibration coefficients.

6.3.2 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*^{3} 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:

$$pHcorr = pHraw + x_0 \cdot (pHraw - x_1) \cdot (value(n_0) - x_2)$$

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):

$$pHcorr = pHraw + x_0 \cdot (pHraw - x_1) \cdot (sub(T) - x_2)$$

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.

Examples

```
>> calibration 4 type
<< calibration 4 type = ph__02
```

Confirm the channel type.

```
>> calibration 4 datetime = 20171201000000, c0 = 15.23461, c1 = -0.198743
```

Set the core coefficients.

```
>> calibration 4 datetime = 20171201000000, x0 = -0.00302, x1 = 7, x2 = 24.943
```

Set the cross-channel correction coefficients.

```
>> calibration 4
<< calibration 4 type = ph__02, datetime = 20171201000000, c0 = 15.23461,c1 = -0.198743,
x0 = -0.00302, x1 = 7, x2 = 24.943, n0 = value
```

Request confirmation of all calibration coefficients.

```
>> settings temperature = 3.9
<< settings temperature = 3.9
```

Set the default fallback temperature value.

6.3.3 Example 3: corr_pres2 - temperature correction of pressure

Returning to the example of an RBR*concerto*^{3} 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:

$$Praw = c_0 + c_1 \cdot R + c_2 \cdot R^2 + c_3 \cdot R^3$$

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:

$$Pcorr = Pcal + \frac{(Praw - Pcal) - Kp_1 \cdot (T - Tcal) - Kp_2 \cdot (T - Tcal)^2 - Kp_3 \cdot (T - Tcal)^3}{1 + Kp_4 \cdot (T - Tcal)}$$

Casting this into the form used by the logger would yield:

$$Pcorr = x_0 + \frac{(Praw - x_0) - x_1 \cdot (value(n_0) - x_5) - x_2 \cdot (value(n_0) - x_5)^2 - x_3 \cdot (value(n_0) - x_5)^3}{1 + x_4 \cdot (value(n_0) - x_5)}$$

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 "T",
- value(**n0**) is the final output value of the temperature channel in °C,
- Pcorr is the corrected output in dbar.

Examples

```
>> calibration 3 type
<< calibration 3 type = pres19
```

Confirm the channel type.

```
>> calibration 3 datetime = 20171201000000, c0 = 0.2346, c1 = 120.9873, c2 = 2. 7356, c3
= 0.7
```

Set the core coefficients.

```
>> calibration 3 datetime = 20171201000000, 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 = 20171201000000, 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 all calibration coefficients.

6.3.4 Example 4: corr_cond - conductivity corrections

Combined CT cell formula revision

In September 2018, RBR Ltd. introduced a new equation for the combined CT cell. Instruments with a combined CT cell manufactured before the 1st September 2018 are using the equation described here. The difference between the two corrections is a move from a first- to third-order polynomial for pressure dependence.

Continuing with the RBR*concerto*^{3} C.T.D.pH logger, the conductivity reading from Channel-1 without corrections is

given by a simple linear function:

$$Craw = c_0 + c_1 \cdot R$$

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:

$$Corr = \frac{Craw - Kc_1 \cdot (Tcond - Tcal)}{1 + Kc_2 \cdot (Tcond - Tcal) + Kc_3 \cdot (Pcorr - Pcal)}$$

Casting the equation into the style used by the logger would give:

$$Corr = \frac{Craw - x_0 \cdot (value(n_0) - x_3)}{1 + x_1 \cdot (value(n_0) - x_3) + x_2 \cdot (value(n_1) - x_4)}$$

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 internal temperature of the conductivity cell channel, in this example 8,
  value(**n0**) is the final output value of the internal temperature of the conductivity cell 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).

Examples

```
>> channel 1 type
<< channel 1 type = cond09
```

Confirm the channel type.

```
>> calibration 1 datetime = 20171201000000, c0 = 0.2346, c1 = 153.4873
```

Set the core coefficients.

```
>> calibration 1 datetime = 20171201000000, 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 label = conductivity_00, datetime = 20171201000000, c0 = 0.2346, c1 =
153.4873, x0 = 0.2003, x1 = 0.2943, x2 = 0.085, x3 = 15.028, x4 = 10.0025, n0 = 8, n1 =
3
```

Request confirmation of all calibration coefficients.

6.3.5 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.

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** command, and the Section Supported channel types). 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*^{3} 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 for details of this temperature channel.

The equation for the temperature corrected output of the DO sensor, as provided by JFE Advantech, is:

$$DOtcomp = \frac{A}{1 + D \cdot (Tr - 25)} + \frac{B}{(N - F)(1 + D(Tr - 25)) + C + F}$$

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:

$$DOtcomp = \frac{x_0}{1 + x_3 \cdot (value(n_0) - 25)} + \frac{x_1}{(N - x_5)(1 + x_3(value(n_0) - 25)) + x_2 + x_5}$$

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:

$$DOcorr = (G + H \cdot DOtcomp) \cdot (1 + E \cdot P)$$

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:

$$DOcorr = (c_0 + c_1 \cdot DOtcomp) \cdot (1 + x_4 \cdot value(n1))$$

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.

Examples

```
>> calibration 4 type
<< calibration 4 type = doxy02
```

Confirm the channel type.

```
>> calibration 4 datetime = 20171201000000, c0 = 0.346, c1 = 1.08873
```

Set the "user" coefficients.

```
>> calibration 4 datetime = 20171201000000, 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 = 20171201000000, 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 all calibration coefficients.

6.3.6 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.

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** command, and the Section Supported channel types). 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*^{3} 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), 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:

$$DOtcomp = \frac{A}{1 + D \cdot (Tr - 25)} + \frac{B}{(N - F)(1 + D \cdot (Tr - 25)) + C + F}$$

as before, but in this case Tr is a "delayed" version of the monitored temperature, computed by the following simple filter operation:

$$Tr(n) = K \cdot T + (1 - K) \cdot Tr(n - 1)$$

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:

$$K = \frac{P}{TCdo - TCt}$$

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:

$$Tr(n) = K \cdot value(n_0) + (1 - K) \cdot Tr(n - 1)$$

In the example discussed here, the value of **n0** is 2, and value(**n0**) is the temperature reported by the logger.

Examples

```
>> calibration 4 type
<< calibration 4 type = doxy06
```

Confirm the channel type.

```
>> calibration 4 datetime = 20171201000000, c0 = 0.346, c1 = 1.08873
```

Set the "user" coefficients.

```
>> calibration 4 datetime = 20171201000000, 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 = 20171201000000, 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 all calibration coefficients.

6.3.7 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*^{3} 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. 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. If the salinity calculation leads to an aberrant value, which happens generally when the conductivity sensor is in the air and reads a slightly negative value, the logger will saturate the salinity value to zero instead of generating an error.

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*^{3} 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".

If the PSS78 calculation generates an error, the datalogger will report a salinity of 0. This might occur when, in air, the conductivity report a small negative value. This does not apply if one of the parameters is already flagged error.

Examples

```
>> calibration 4 type
<< calibration 4 type = sal_00
```

Confirm the channel type.

```
>> calibration 4
<< calibration 4 type = sal_00, datetime = 20171201000000, n0 = 3, n1 = 2, n2 = 1, n3 =
value
```

Request confirmation of all calibration coefficients.

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 = 20171201000000, n0 = 2, n1 = value, n2 = 1,
n3 = value
```

Request confirmation of all calibration coefficients.

6.3.8 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*^{3} 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".

Examples

```
>> calibration 4 type
<< calibration 4 type = pres08
```

Confirm the channel type.

```
>> calibration 4
<< calibration 4 type = pres08, datetime = 20171201000000, n0 = 3, n1 = value
```

Request confirmation of all calibration coefficients.

6.3.9 Example 9: depth - derivation of depth from pressure

This derived channel implements a simplified equation for water depth in metres, 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.

$$Dm = \frac{P - Patm}{p \cdot g}$$

In the form used by the logger, using an RBR*concerto*^{3} C.T.D as an example, this becomes:

$$Dm = \frac{value(n_0) - value(n_1)}{p \cdot g}$$

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".
- **p** 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 metres.

Examples

```
>> calibration 4 type
<< calibration 4 type = dpth01
```

Confirm the channel type.

```
>> calibration 4
<< calibration 4 type = dpth01, datetime = 20171201000000, n0 = 3, n1 = value
```

Request confirmation of all calibration coefficients.

```
>> settings density
<< settings density = 1.026021
>> settings density = 1.0197
<< settings density = 1.0197
```

Request, then change, the programmed water density.

6.3.10 Example 10: corr_metsmeth - temperature correction of METS methane output

An RBR*concerto*^{3} 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) for details of this temperature channel.

The equation for the temperature corrected concentration output of the methane sensor, **Cm**, as provided by Franatech, is:

$$Cm = exp(c_0 \cdot ln((c_1 + c_2 \cdot exp(\frac{-Vt}{c_3})) \cdot (\frac{1}{Vm} - \frac{1}{c_4 + c_5 \cdot exp(\frac{-Vt}{c_6})})))$$

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.

$$Vm = x_0 + x_1 \cdot R$$

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:

$$Vt = \frac{Tm - ct_0}{ct_1}$$

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**).

Examples

```
>> calibration 4 type
<< calibration 4 type = meth00
```

Confirm the channel type.

```
>> calibration 4 datetime = 20171201000000, 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 = 20171201000000, x0 = 6.782656, x1 = -9.257345
```

Set the secondary coefficients for the voltage conversion.

```
>> calibration 4
<< calibration 4 type = meth00, datetime = 20171201000000, 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 the primary methane channel.

```
>> calibration 5 c0 c1
<< calibration 5 c0 = -5.65608, c1 = 16.80047
```

Confirm the main coefficients of the temperature compensation channel.

6.3.11 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.

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** command, and the Section Supported channel types). 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*^{3} 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 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:

$$DOtcomp = \frac{A}{1 + D \cdot (Tr - 25) + F \cdot (Tr - 25)^2} + \frac{B}{N \cdot (1 + D \cdot (Tr - 25) + F \cdot (Tr - 25)^2) + C}$$

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:

$$DOtcomp = \frac{x_0}{1 + x_3 \cdot (value(n_0) - 25) + x_5 \cdot (value(n_0) - 25)^2} + \frac{x_1}{N \cdot (1 + x_3 \cdot (value(n_0) - 25) + x_5 \cdot (value(n_0) - 25)^2) + x_2}$$

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:

$$DOcorr = (G + H \cdot DOtcomp) \cdot (1 + E \cdot P)$$

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:

$$DOcorr = (c_0 + c_1 \cdot DOtcomp) \cdot (1 + x_4 \cdot value(n_1))$$

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.

An RBR*concerto*^{3} 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) for details of this temperature channel.

The equation for the temperature corrected concentration output of the methane sensor, **Cm**, as provided by Franatech, is:

$$Cm = exp(c_0 \cdot ln((c_1 + c_2 \cdot exp(\frac{-Vt}{c_3})) \cdot (\frac{1}{Vm} - \frac{1}{c_4 + c_5 \cdot exp(\frac{-Vt}{c_6})})))$$

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.

$$Vm = x_0 + x_1 \cdot R$$

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:

$$Vt = \frac{Tm - ct_0}{ct_1}$$

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**).

Examples

```
>> calibration 4 type
<< calibration 4 type = doxy08
```

Confirm the channel type.

```
>> calibration 4 datetime = 20171201000000, c0 = 0.346, c1 = 1.08873
```

Set the "user" coefficients.

```
>> calibration 4 datetime = 20171201000000, 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 = 20171201000000, 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 all calibration coefficients.

6.3.12 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.

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** command, and the Section

Supported channel types). 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*^{3} 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), 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:

$$DOtcomp = \frac{A}{1 + D \cdot (Tr - 25) + F \cdot (Tr - 25)^2} + \frac{B}{N \times (1 + D \cdot (Tr - 25) + F \cdot (Tr - 25)^2) + C}$$

as before, but in this case Tr is a 'delayed' version of the monitored temperature, computed by the following simple filter operation:

$$Tr(n) = K \cdot T + (1 - K) \cdot Tr(n - 1)$$

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:

$$K = \frac{P}{TCdo - TCt}$$

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:

$$Tr(n) = K \cdot value(n_0) + (1 - K) \cdot Tr(n - 1)$$

In the example discussed here, the value of **n0** is 2, and value(**n0**) is the temperature reported by the logger.

After calculating DOtcomp using this 'delayed' temperature value, it then becomes the input to a simple linear equation, which according to JFE Advantech can be further compensated for pressure effects as follows:

$$DOcorr = (G + H \cdot DOtcomp) \cdot (1 + E \cdot P)$$

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:

$$DOcorr = (c_0 + c_1 \cdot DOtcomp) \cdot (1 + x_4 \cdot value(n_1))$$

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.

Examples

```
>> calibration 4 type
<< calibration 4 type = doxy09
```

Confirm the channel type.

```
>> calibration 4 datetime = 20171201000000, c0 = 0.346, c1 = 1.08873
```

Set the "user" coefficients.

```
>> calibration 4 datetime = 20171201000000, 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 = 20171201000000, 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 all calibration coefficients.

6.3.13 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*^{3} 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.

Examples

```
>> calibration 7 type
<< calibration 7 type = sos_00
```

Confirm the channel type.

```
>> calibration 7
<< calibration 7 type = sos_00, datetime = 20171201000000, n0 = 3, n1 = 4, n2 = 6
```

Request confirmation of all channel indices.

6.3.14 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 µS/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.

$$C25 = Ccorr \cdot \frac{K_0}{1 + K_1 \cdot (T - 25)}$$

where

- **Ccorr** is the standard conductivity reading (already compensated for temperature dependence of the measurement circuit as described in Example 4),
- **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 µS/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** command, using the **speccondtempco** parameter. If this parameter is never explicitly set, the default value used is 0.0191.

6.3.15 Example 15: deri_bprpres and deri_bprtemp, BPR channels

Loggers with BPR channels interface a Paroscientific, Inc. transducer. The logger measures 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 (**peri00** and **peri01**). 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 types of equations: **deri_bprpres** (pressure in dbar) and **deri_bprtemp** (temperature in °C).

deri_bprpres equation

$$\begin{aligned}\text{Pressure (dbar)} &= (C \cdot (1 - (\frac{T_0}{T})^2)) \cdot (1 - D(1 - (\frac{T_0}{T})^2)) \cdot 0.689475728 \\ T &= value(n_0) \cdot 1e-6 \\ X &= value(n_1) \cdot 1e-6 \\ U &= X - x_0 \\ C &= x_1 + x_2 \cdot U + x_3 \times U^2 \\ D &= x_4 + x_5 \cdot U \\ T_0 &= x_6 + x_7 \cdot U + x_8 \cdot U^2 + x_9 \cdot U^3 + x_{10} \cdot U^4\end{aligned}$$

- The coefficients **x0 ... x10** are provided by Paroscientific, Inc. and are mapped according 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

$$\begin{aligned}\text{Temperature} &= x_1 \cdot U + x_2 \cdot U^2 + x_3 \cdot U^3 \\ X &= value(n_0) \cdot 1e-6 \\ U &= X - x_0\end{aligned}$$

- The coefficients **x0 ... x3** are provided by Paroscientific, Inc. and are mapped according 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.

Examples

```
>> calibration 3 type
<< calibration 3 type = bpr_08
>> calibration 4 type
<< calibration 4 type = bpr_09
```

Confirm the channel type.

```
>> calibration 3 datetime= 20171123120721, 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 = 20171123120721, 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
>> calibration 3 datetime = 20171123120721, x8 = 58.803409E+00, x9 = 180.91160E+00, x10
= 0.0000000E+00
<< calibration 3 datetime = 20171123120721, x8 = 58.803408e+000, x9 = 180.91160e+000,
x10 = 0.0000000e+000
```

Set the pressure coefficients provided by Paroscientific, Inc.

```
>> calibration 4 datetime = 20171123120722, x0 = 5.8310298E+00, x1 = -3.8981210E+03, x2
= -10.493120E+03, x3 = 0.0000000E+00
<< calibration 4 datetime = 20171123120722, x0 = 5.8310300e+000, x1 = -3.8981210e+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 = 20171123120721, 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 = 58.803408e+000, x9 =
180.91160e+000, x10 = 0.0000000e+000, n0 = 1, n1 = 2
>> calibration 4
<< calibration 4 type = bpr_09, datetime = 20171123120722, x0 = 5.8310300e+000, x1 =
-3.8981210e+003, x2 = -10.493120e+003, x3 = 0.0000000e+000, n0 = 2
```

Request confirmation of all pressure and temperature calibration coefficients.

6.3.16 Example 16: distancefromechotiming distance from echo timing

An RBR*concerto*^{3} Alti has a RBRalti sensor which outputs distance and echo timing. Only distance is reported by the RBR*concerto*^{3}.

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 follows:

$$\text{Distance} = \frac{Te}{1000} \cdot \frac{ASS}{2.0}$$

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**").

Examples

```
>> calibration 1 type
<< calibration 1 type = alti00
```

Confirm the channel type.

```
>> calibration 1
<< calibration 1 type = alti00, datetime = 20171201000000, n0 = 2, n1 = value
```

Request confirmation of all calibration coefficients.

6.3.17 Example 17: corr_o2conc_garcia, O2 concentration compensated for salinity and pressure

Consider an RBR*concerto*^{3} C.T.D.DO logger, where the DO channel is a RBR*coda* ODO. The RBR*coda* ODO transfers both the foil temperature and the dissolved oxygen concentration (not compensated for salinity). The RBR*concerto*^{3} calculates the concentration compensated for salinity and pressure first:

$$O2corr = (C_0 + C_1 \cdot O2unc) \cdot exp(S \cdot (Gb_0 + Gb_1 \cdot Ts + Gb_2 \cdot Ts^2 + Gb_3 \cdot Ts^3) + Gc_0 \cdot S^2) \cdot (1 + P \cdot C_2)$$

where

- O2corr is the corrected O2 concentration, compensated for salinity and pressure,
- O2unc is the uncompensated O2 concentration returned by the RBR*optode*,
- C_0 and C_1 are corrections and scaling factors for the uncompensated O2 concentration,
- C_2 is a correction factor for pressure,
- S is the salinity in PSU,
- P is sea pressure in dbar,

and

$$Ts = ln(\frac{298.15 - T}{273.15 + T})$$

where T is the water temperature (in °C), and

$$Gb_0 = -6.24523e-3$$
$$Gb_1 = -7.37614e-3$$
$$Gb_2 = -1.03410e-2$$
$$Gb_3 = -8.17083e-3$$
$$Gc_0 = -4.88682e-7$$

which correspond to Garcia and Gordon coefficients.

The corresponding logger coefficients are:

- **c0** is C_0
- **c1** is C_1
- **c2** is C_2
- **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 the atmospheric pressure channel (set to **value** in order to use **settings atmosphere**)

One might change **c0** and **c1** in order to perform a two-point calibration to the RBR*coda* ODO. Please note that those coefficients are used and stored only by the logger, not the RBR*coda* ODO.

Examples

```
>> calibration 4 type
<< calibration 4 type = doxy23
```

Confirm the channel type.

```
>> calibration 4
<< calibration 4 type = doxy23, datetime = 20171201000000, c0 = 0,c1 = 1, c2 = 3.25e-5,
n0 = 5, n1 = 8, n2 = 3, n3 = value
```

Request confirmation of all calibration coefficients.

6.3.18 Example 18: deri_o2sat_garcia, derived O2 saturation from concentration

Consider an RBR*concerto*^{3} C.T.D.DO logger, where the DO channel is a RBR*coda* ODO. The RBR*coda* ODO transfers both the foil temperature and the dissolved oxygen concentration (not compensated for salinity). The RBR*concerto*^{3} calculates the concentration compensated for salinity first, then the air saturation (in %) via the following equation:

$$\text{Air saturation} = 2.23916 \cdot \frac{(10.1325 - Pav) \cdot Cc}{(Patm - Pav) \cdot \text{Solubility}}$$

where

- Cc is the concentration in uMol/L, compensated for salinity,
- Patm is atmospheric pressure, and
- Pav is air vapour pressure.

Solubility is calculated via Gordon and Garcia as:

$$\text{Solubility} = exp((Ga_0 + Ga_1 \cdot Ts + Ga_2 \cdot Ts^2 + Ga_3 \cdot Ts^3 + Ga_4 \cdot Ts^4 + Ga_5 \cdot Ts^5)$$
$$+S \cdot (Gb_0 + Gb_1 \cdot Ts + Gb_2 \cdot Ts^2 + Gb_3 \cdot Ts^3) + Gc_0 \cdot S^2)$$

where S is the salinity in PSU and Ts is defined as:

$$Ts = ln(\frac{298.15 - T}{273.15 + T})$$

with T being the water temperature in °C and:

$$\begin{aligned} Ga_0 &= 2.00856 \\ Ga_1 &= 3.22400 \\ Ga_2 &= 3.99063 \\ Ga_3 &= 4.80299 \\ Ga_4 &= 9.78188e-1 \\ Ga_5 &= 1.71069 \\ Gb_0 &= -6.24523e-3 \\ Gb_1 &= -7.37614e-3 \\ Gb_2 &= -1.03410e-2 \\ Gb_3 &= -8.17083e-3 \\ Gc_0 &= -4.88682e-7 \end{aligned}$$

Air vapour pressure (in dbar) is calculated as:

$$Pav = \frac{exp(52.57 - \frac{6690.9}{T + 273.15} - 4.6818 \cdot ln(T + 273.15))}{100}$$

The corresponding logger coefficients are:

- **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 (set to **value** in order to use **settings atmosphere**)

The O_2 saturation equation was revised in firmware 1.094 of the RBR*concerto*[3], RBR*maestro*[3], RBR*duo*[3], RBR*legato*[3] and RBR*argo*[3]. For the previous implementation of this equation, refer to the version B of this document.

Examples

```
>> calibration 9 type
<< calibration 9 type = doxy22
```

Confirm the channel type.

```
>> calibration 9
<< calibration 9 type = doxy22, datetime = 20171201000000, n0 = 4, n1 = 5, n2 = 8, n3 =
value
```

Request confirmation of all channel indices.

6.3.19 Example 19: corr_cond1 - conductivity corrections for deep CT cell

Continuing with the RBR*concerto*[3] C.T.D.pH logger, the conductivity reading from Channel-1 without corrections is given by a simple linear function:

$$Craw = c_0 + c_1 \cdot R$$

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:

$$Corr = \frac{Craw - Kc_1 \cdot (Tcond - Tcal)}{1 + Kc_2 \cdot (Tcond - Tcal) + (Kp_1 \cdot (Pcorr - Pcal) + Kp_2 \cdot (Pcorr - Pcal)^2 + Kp_3 \cdot (Pcorr - Pcal)^3 + Kp_4 \cdot (Pcorr - Pcal)^{Kp_5})}$$

Casting the equation into the style used by the logger would give:

$$Corr = \frac{Craw - x_0 \cdot (value(n_0) - x_7)}{1 + x_1 \cdot (value(n_0) - x_7) + (x_2 \cdot (value(n_1) - x_8) + x_3 \cdot (value(n_1) - x_8)^2 + x_4 \cdot (value(n_1) - x_8)^3 + x_5 \cdot (value(n_1) - x_8)^{x_6})}$$

where

- Craw is the uncorrected conductivity, **c0** + **c1** × **R,**
- **x0**, **x1** correspond respectively to the temperature compensation constants "Kc1", "Kc2"
- **x2, x3, x4, x5, x6** correspond respectively to the pressure compensation constants "Kp1", "Kp2", "Kp3", "Kp4", "Kp5"
- **x7** is the calibration temperature "Tcal" in °C,
- **x8** is the calibration pressure "Pcal" in dbar,
- **n0** is the index of the internal temperature of the conductivity cell channel, in this example 8, value(**n0**) is the final output value of the internal temperature of the conductivity cell 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).

Examples

```
>> channel 1 type
<< channel 1 type = cond11
```

Confirm the channel type.

```
>> calibration 1 datetime = 20171201000000, c0 = 0.2346, c1 = 153.4873
```

Set the core coefficients.

```
>> calibration 1 datetime = 20171201000000, x0 = 0.2003, x1 = 0.2943, x2 = 0.005, x3 =
0.085, x4 = 0.0001, x5 = 0.0000, x6= 0.0000, x7 = 15.028, x8 = 10.0025
```

Set the cross-channel correction coefficients.

```
>> calibration 1
<< calibration 1 label = conductivity_00, datetime = 20171201000000, c0 = 0.2346, c1 =
153.4873, x0 = 0.2003, x1 = 0.2943, x2 = 0.005, x3 = 0.085, x4 = 0.0001, x5 = 0.0000,
x6= 0.0000, x7 = 15.028, x8 = 10.0025, n0 = 8, n1 = 3
```

Request confirmation of all calibration coefficients.

6.3.20 Example 20: corr_cond2 - conductivity corrections for CT cell

> **CT cell formula revision**
>
> In September 2018, RBR Ltd. introduced a new equation for the combined CT cell. Instruments manufactured after the 1st September 2018 are using this equation. The difference between the two corrections is a move from a first- to third-order polynomial for pressure dependence.

Continuing with the RBR*concerto*^{3} C.T.D.pH logger, the conductivity reading from Channel-1 without corrections is given by a simple linear function:

$$Craw = c_0 + c_1 \cdot R$$

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:

$$Corr = \frac{Craw - Kc_1 \cdot (Tcond - Tcal)}{1 + Kc_2 \cdot (Tcond - Tcal) + (Kp_1 \cdot (Pcorr - Pcal) + Kp_2 \cdot (Pcorr - Pcal)^2 + Kp_3 \cdot (Pcorr - Pcal)^3)}$$

Casting the equation into the style used by the logger would give:

$$Corr = \frac{Craw - x_0 \cdot (value(n_0) - x_5)}{1 + x_1 \cdot (value(n_0) - x_5) + (x_2 \cdot (value(n_1) - x_6) + x_3 \cdot (value(n_1) - x_6)^2 + x_4 \cdot (value(n_1) - x_6)^3)}$$

- Craw is the uncorrected conductivity, **c0** + **c1** × **R,**
- **x0**, **x1** correspond respectively to the temperature compensation constants "Kc1", "Kc2"
- **x2, x3, x4** correspond respectively to the pressure compensation constants "Kp1", "Kp2", "Kp3"
- **x5** is the calibration temperature "Tcal" in °C,
- **x6** is the calibration pressure "Pcal" in dbar,
- **n0** is the index of the internal temperature of the conductivity cell channel, in this example 8,
  value(**n0**) is the final output value of the internal temperature of the conductivity cell 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).

Examples

```
>> channel 1 type
<< channel 1 type = cond12
```

Confirm the channel type.

```
>> calibration 1 datetime = 20171201000000, c0 = 0.2346, c1 = 153.4873
```

Set the core coefficients.

```
>> calibration 1 datetime = 20171201000000, x0 = 0.2003, x1 = 0.2943, x2 = 0.005, x3 =
0.085, x4 = 0.0001, x5 = 15.028, x6 = 10.0025
```

Set the cross-channel correction coefficients.

```
>> calibration 1
<< calibration 1 label = conductivity_00, datetime = 20171201000000, c0 = 0.2346, c1 =
153.4873, x0 = 0.2003, x1 = 0.2943, x2 = 0.005, x3 = 0.085, x4 = 0.0001, x5 = 15.028, x6
= 10.0025, n0 = 8, n1 = 3
```

Request confirmation of all calibration coefficients.

6.3.21 Example 21: corr_cond3 - conductivity corrections for RBRLegato and 6000dbar C and CT cell

With the RBR*legato*^{3} C.T.D logger, the conductivity reading from the conductivity channel without corrections is given by a simple linear function:

$$Craw = c_0 + c_1 \cdot c_2 \cdot R$$

where R is the normalized voltage ratio from the channel monitoring conductivity, **c0,c1** are the core coefficients of the linear equation, along with **c2** which is a geometrical factor (K-factor) reflecting the final installation of conductivity cell, 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:

$$Corr = \frac{Craw - Kc_1 \cdot (Tcond - Tcal)}{1 + Kc_2 \cdot (Tcond - Tcal) + (Kp_1 \cdot (Pcorr - Pcal) + Kp_2 \cdot (Pcorr - Pcal)^2 + Kp_3 \cdot (Pcorr - Pcal)^3)}$$

Casting the equation into the style used by the logger would give:

$$Corr = \frac{Craw - x_0 \cdot (value(n_0) - x_5)}{1 + x_1 \cdot (value(n_0) - x_5) + (x_2 \cdot (value(n_1) - x_6) + x_3 \cdot (value(n_1) - x_6)^2 + x_4 \cdot (value(n_1) - x_6)^3)}$$

- Craw is the uncorrected conductivity, **c0** + **c1** × **c2** × **R.**
- **x0**, **x1** correspond respectively to the temperature compensation constants "Kc1", "Kc2".
- **x2, x3, x4** correspond respectively to the pressure compensation constants "Kp1", "Kp2", "Kp3".
- **x5** is the calibration temperature "Tcal" in °C.
- **x6** is the calibration pressure "Pcal" in dbar.
- **n0** is the index of the internal temperature of the conductivity cell channel, in this example 7, value(**n0**) is the final output value of the internal temperature of the conductivity cell 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).

Examples

```
>> channel 1 type
<< channel 1 type = cond13
```

Confirm the channel type.

```
>> calibration 1 datetime = 20171201000000, c0 = 0.2346, c1 = 153.4873, c2 = 1.0001
```

Set the core coefficients.

```
>> calibration 1 datetime = 20171201000000, x0 = 0.2003, x1 = 0.2943, x2 = 0.0051, x3 =
0.085, x4 = 0.0001, x5 = 15.0280, x6 = 10.0025
```

Set the cross-channel correction coefficients.

```
>> calibration 1
<< calibration 1 label = conductivity_00, datetime = 20171201000000, c0 = 0.2346, c1 =
153.4873, c2 = 1.0001 x0 = 0.2003, x1 = 0.2943, x2 = 0.0051, x3 = 0.0850, x4 = 0.0001,
x5 = 15.0280, x6 = 10.0025, n0 = 7, n1 = 3
```

Request confirmation of all calibration coefficients.

6.3.22 Example 22: corr_pres5 - temperature correction of pressure

This equation computes a temperature corrected pressure value, with the temperature, T_{sensor}, derived from a voltage ratio corresponding to temperature of the sensor itself R_t, as follows, corrected by the pressure ratio signal *R*:

$$T_{sensor} = \sum_{n=0}^{3} K_{t_n} (R_T - \sum_{m=0}^{3} Kc_m (A \cdot R + B)^m)^n$$

Where the Kt terms are the polynomial coefficients for the temperature calculation, the Kc terms are the polynomial coefficients of the pressure correction, while A and B are coefficients of the measurement circuit linear model.

Once the temperature is known, we proceed almost exactly as in example 3 above, so the pressure reading, without correction for the effect of temperature on the sensor, is given by a cubic polynomial:

$$Praw = c_0 + c_1 \cdot R + c_2 \cdot R^2 + c_3 \cdot R^3$$

where **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:

$$Pcorr = Pcal + \frac{(Praw - Pcal) - Kp_1 \cdot (T - Tcal) - Kp_2 \cdot (T - Tcal)^2 - Kp_3 \cdot (T - Tcal)^3}{1 + Kp_4 \cdot (T - Tcal)}$$

Casting this into the form used by the logger would yield:

$$Pcorr = x_0 + \frac{(Praw - x_0) - x_1 \cdot (T_{sensor} - x_5) - x_2 \cdot (T_{sensor} - x_5)^2 - x_3 \cdot (T_{sensor} - x_5)^3}{1 + x_4 \cdot (T_{sensor} - x_5)}$$

where

- *Praw* is the cubic polynomial in *R* as in example 3.
- **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.
- **x6,x7** correspond to the circuit gain parameters A and B.
- **x8**,**x9**,**x10**,**x11** correspond to the constants "Kc0" through "Kc3".
- **x12**,**x13**,**x14**,**x15** correspond to the constants "Kt1", through "Kt3" used to compute the sensor temperature.
- **n0** is the index of the sensor temperature ratio channel.
- T_{sensor} is the computed sensor temperature in °C.
- *Pcorr* is the corrected output in dbar.

Examples

```
>> calibration 3 type
<< calibration 3 type = pres28
```

Confirm the channel type.

```
>> calibration 3 datetime = 20171201000000, c0 = 0.2346, c1 = 120.9873, c2 = 2. 7356, c3
= 0.7
```

Set the core coefficients.

```
>> calibration 3 datetime = 20171201000000, x0 = 9.983, x1 = 0.2003, x2 = 0.2943, x3 =
0.0721, x4 = 0.1049, x5 = 21.29
```

```
>> calibration 3 datetime = 20171201000000, x6 = 0, x7 = 187.06043
```

```
>> calibration 3 datetime = 20171201000000, x8 = 14.219203, x9 = -0.30655386, x10 =
0.0012941403, x11 =0.0000039787190
```

```
>> calibration 3 datetime = 20171201000000, x12 = -3727.6430, x13 = 2.8672020, x14 =
-0.00075268429, x15 = 0.000000068588312
```

```
>> calibration 3 datetime = 20171201000000, n0 = 5
```

Set the cross-channel correction coefficients.

```
>> calibration 3
<< calibration 3 type = pres28, datetime = 20171201000000, 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, x6 = 0.0, x7 = 187.06043e+000, x8= 14.219203, x9
= -306.55386e-003, x10 = 1.2941403e-003, x11 = 3.9787190e-006, x12 = -3.7276430e+003,
x13 = 2.8672020e+000, x14 = -752.68429e-006, x15 = 68.588312e-009, n0 = 5
```

Request confirmation of all calibration coefficients.

6.3.23 Example 23: corr_irr - irradiance

This equation computes first a temperature corrected irradiance value in Volts:

$$Irr_{tempcompensated} = x_2 \cdot (x_0 + x_1 \cdot R) + (x_3 + x_4 \cdot T_{sensor} + x_5 \cdot T_{sensor}^2)$$

Then applies a linear equation to convert to final units:

$$Irr_{calibrated} = c_0 + c_1 \cdot Irr_{tempcompensated}$$

where

- **c0, c1** are calibration coefficients.
- **x0**, **x1, x2** are coefficients required to convert the raw data in Volts.
- **x3, x4, x5** are the temperature correction coefficients.
- **n0** is the index of the temperature of the sensor channel.
- T_{sensor} is the temperature of the sensor in °C.
- $Irr_{tempcompensated}$ is the measured irradiance in Volts and compensated in temperature.
- $Irr_{calibrated}$ is the measured irradiance in final calibration units.

Examples

```
>> channel 3 type
<< channel 3 type = irr_01
```

Confirm the channel type.

```
>> calibration 3 datetime = 20171201000000, c0 = 0.000, c1 = 1.0000
```

Set the core coefficients.

```
>> calibration 3 datetime = 20171201000000, x0 = 9.983, x1 = 0.2003, x2 = 1.000, x3 =
0.0721, x4 = 0.1049, x5 = 21.29
```

Set the cross-channel correction coefficients.

```
>> calibration 3
<< calibration 3 type = irr_01, datetime = 20171201000000, c0 = 0.0000, c1 = 1.0000, x0
= 9.983, x1 = 0.2003, x2 = 1.000, x3 = 0.0721, x4 = 0.1049, x5 = 21.29, n0 = 7
```

Request confirmation of all calibration coefficients.

6.3.24 Example 24: corr_irr2 - generic irradiance and PAR

This equation is a refined and simplified version of the one given in Example 23, and will typically be used to present irradiance in µW/cm²/nm, or PAR in µmol/m²/s.

$$Irr_{tempcompensated} = c_0 + c_1 * c_2 \left(R + x_0 + x_1 * (T_{sensor} - x_3) + x_2 * (T_{sensor} - x_3)^2\right)$$

Then applies a linear equation to convert to final units:

where

- **c0** is a user-adjustable dark offset value.
- **c1** is the primary slope coefficient which transforms the value to final units.
- **c2** is an immersion factor, which corrects for use in a medium other than was used for calibration; water compared with air, for example.
- **R** is the raw voltage ratio measured by the logger.
- **x0**, **x1, x2** are factory-determined coefficients of a quadratic term, used to correct the sensor's dark current for

temperature effects.

- **x3** is the temperature at which the optical calibration was performed.
- T_{sensor} is the internally measured temperature of the sensor in °C.
- $Irr_{tempcompensated}$ is the final value of irradiance or PAR, temperature compensated and presented in final calibration units.

Examples

```
>> channel 3 type
<< channel 3 type = irr_05
```

Confirm the channel type.

```
>> calibration 3 datetime = 20210901085500, c0 = 0.000, c1 = 5079.0, c2 = 1.2
```

Set the core coefficients.

```
>> calibration 3
<< calibration 3 label = irradiance_00, datetime = 20210901085500, c0 = 0.000, c1 =
5079.0, c2 = 1.2, x0 = 0.098, x1 = 0.023, x2 = 0.00037, x3 = 23.4, n0 = 4
```

Request confirmation of all calibration coefficients.

6.3.25 Example 25: deri_dyncorrT and deri_dyncorrS dynamic correction channels

There are two types of dynamic errors affecting salinity measurements: response time and sensor misalignments, and thermal mass errors.

"Response time and sensor misalignments", or "C-T lag", refers to the time lag between temperature and conductivity measurements, which would result in "salinity spiking". It is generated by two separate mechanisms: the physical separation between the thermistor and the conductivity cell, and the inherent response time of the thermistor.

"Thermal mass errors" refers to the thermal mass of the conductivity cell impacting the temperature of the water column where the seawater conductivity is measured. It occurs when the CTD travels through a temperature gradient. Thermal mass errors present two main separate timescales: a long-term thermal mass error (timescales of minutes) and a short-term thermal mass error (timescales of seconds).

In order to correct all those dynamic errors, the logger provides two channels (**temp38** and **sal_01**) that implement the dynamic correction equations (supported since firmware 1.144) . They rely on two types of equations: **deri_dyncorrT** (temperature in °C) and **deri_dyncorrS** (salinity in PSU).

The C-T lag correction and the short-term thermal mass correction are only applied when the sample is acquired while the instrument is actively logging at a sampling rate of at least 1Hz.

The long-term thermal mass correction is only applied when the sample is acquired while the instrument is actively logging at a sampling rate of at least 0.1Hz.

deri_dyncorrT equation

If the sample is acquired or fetched while actively logging at a sampling rate greater or equal to 1Hz:

$$T_{cor}(n) = (1-\phi)\cdot(T_{meas}(n+N)) + \phi \cdot T_{meas}(n+N+1)$$
$$\phi = F_s \cdot \left[\Delta t \mod \frac{1}{F_s}\right]$$
$$N = \lfloor F_s \cdot \Delta t \rfloor$$

Otherwise:

$$T_{cor}(n) = T_{meas}(n)$$

where:

- T_{cor} is the C-T lag corrected temperature (°C)
- T_{meas} is the marine temperature (°C)
- F_s is the sampling frequency (Hz).
- Δt is the C-T lag correction (s).

deri_dyncorrT coefficients name	coefficient name implemented
Δt	**x0**

- **n0** is the index of the supporting marine temperature channel (n0=3 in examples below).. value(**n0**) is the final output value of this channel in degree Celsius, i.e $\mathbf{T_{meas}}$.

deri_dyncorrS equation

$$\text{Salinity(PSU)} = pss78(C, T_{cell}, P)$$
$$T_{cell} = T_{cor} + T_{long} - T_{short}$$

with

$$T_{long} = \text{ctcoeff}(n)\cdot(T_{cond} - T_{cor})$$

and

$$T_{short}(n) = -b(n)\cdot T_{short}(n-1) + a(n)\cdot(T_{cor}(n) - T_{cor}(n-1))$$
$$a(n) = \frac{4f_N\cdot\alpha(n)\cdot\tau(n)}{1+4f_N\cdot\tau(n)}$$
$$b(n) = 1 - 2a(n)\cdot\alpha(n)^{-1}$$
$$f_N = \frac{F_s}{2}$$

where

$$\text{ctcoeff}(n) = \text{ctcoeff}_a * V_p(n)^{\text{ctcoeff}_e}$$
$$\alpha(n) = \alpha_a * V_p(n)^{\alpha_e}$$
$$\tau(n) = \tau_a * V_p(n)^{\tau_e}$$

and

$$V_{p_{est}}(n) = (1-a) * V_{p_{est}}(n-1) + a * (P(n-1) - P(n)) * F_s$$
$$a = 1 - e^{-2*pi*V_{p_{fc}}/F_s}$$
$$V_p(n) = \min\{\max\{V_{p_{est}}(n), V_{p_{min}}\}, V_{p_{max}}\}$$

If the sample is fetched or not acquired while actively sampling at a rate faster or equal to 1Hz:

$$T_{short}(n) = 0$$

If the sample is fetched or not acquired while actively sampling at a rate faster or equal to 0.1Hz:

$$T_{long}(n) = 0$$

where:

- f_N is the Nyquist frequency, defined as half the sampling rate F_s.
- α(n) is the magnitude of short-term thermal mass correction and depends on the instantaneous ascent rate.
- τ(n) is the time constant of short-term thermal mass correction (s) and depends on the instantaneous ascent rate.
- ctcoeff(n) is the magnitude of the long-term thermal mass correction and depends on the instantaneous ascent rate.

deri_dyncorrS coefficients name	coefficient name implemented
α_a	**x0**
α_e	**x1**
τ_a	**x2**
τ_e	**x3**
$ctcoeff_a$	**x4**
$ctcoeff_e$	**x5**
Vp_{min}	**x6**
Vp_{max}	**x7**
Vp_{fc}	**x8**

- **n0** is the index of the conductivity channel. (n0=1 in examples below).
  value(**n0**) is the final output value of this channel in mS/cm, i.e., **C**.
- **n1** is the index of the sea pressure channel. (n1=2 in examples below).
  value(**n1**) is the final output value of this channel in dbar i.e., **P**.
- **n2** is the index of the C-T lag corrected temperature channel. (n2=3 in examples below).
  value(**n2**) is the final output value of this channel in °C, i.e., $\mathbf{T_{cor}}$.
- **n3** is the index of the internal temperature of the conductivity cell channel. (n3=7 in examples below).
  value(**n3**) is the final output value of this channel in °C, i.e., $\mathbf{T_{cond}}$.

Examples

```
>> calibration 5 type
<< calibration 5 type = temp38
>> calibration 6 type
<< calibration 6 type = sal_01
```

Confirm the channel type.

```
>> calibration 5 datetime= 20220119163000, x0 = 0.3500
<< calibration 5 datetime= 20220119163000, x0 = 0.3500
```

Set the temperature coefficients.

```
>> calibration 6 datetime = 20220119163000, x0=0.1300, x1=5.9000, x2 = 1.0200e-00
<< calibration 6 datetime = 20220119163000, x0=0.1300, x1=5.9000, x2 = 1.0200e-00
```

Set the salinity coefficients provided.

```
>> calibration 5
<< calibration 5 type = temp38, datetime = 20220119163000, x0 = 0.3500, n0=3
>> calibration 6
<< calibration 6 type = sal_01, datetime = 20220119163000, x0 = 0.3700, x1 = -1.0300, x2
= 16.0200, x3 = -0.2600, x4 = 0.1400, x5 = -1.0000, x6 = 0.0400, x7 = 0.0500, x8 =
0.1500, n0=1, n1=2, n2=3, n3=7
```

Request confirmation of all temperature and salinity calibration coefficients.

6.3.26 Example 26: corr_cond4 - conductivity corrections for RBRLegato and 6000dbar C and CT cell

With the RBR*legato*[3] C.T.D logger, the conductivity reading from the conductivity channel without corrections is given by a simple linear function:

$$Craw = c_0 + c_1 \cdot c_2 \cdot R$$

where R is the normalized voltage ratio from the channel monitoring conductivity, **c0,c1** are the core coefficients of the linear equation, along with **c2** which is a geometrical factor (K-factor) reflecting the final installation of conductivity cell, 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:

$$Corr = \frac{Craw - Kc_1 \cdot (Tcond - Tcal)}{1 + Kc_2 \cdot (Tcond - Tcal) + (Kp_1 \cdot (Pcorr - Pcal) + Kp_2 \cdot (Pcorr - Pcal)^2 + Kp_3 \cdot (Pcorr - Pcal)^3 + Kp_4 \cdot (Pcorr - Pcal)^4)}$$

Casting the equation into the style used by the logger would give:

$$Corr = \frac{Craw - x_0 \cdot (value(n_0) - x_6)}{1 + x_1 \cdot (value(n_0) - x_6) + (x_2 \cdot (value(n_1) - x_7) + x_3 \cdot (value(n_1) - x_7)^2 + x_4 \cdot (value(n_1) - x_7)^3 + x_5 \cdot (value(n_1) - x_7)^4)}$$

- Craw is the uncorrected conductivity, **c0** + **c1** × **c2** × **R.**
- **x0**, **x1** correspond respectively to the temperature compensation constants "Kc1", "Kc2".
- **x2, x3, x4, x5** correspond respectively to the pressure compensation constants "Kp1", "Kp2", "Kp3", "Kp4".
- **x6** is the calibration temperature "Tcal" in °C.
- **x7** is the calibration pressure "Pcal" in dbar.
- **n0** is the index of the internal temperature of the conductivity cell channel, in this example 7, value(**n0**) is the final output value of the internal temperature of the conductivity cell 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).

Examples

```
>> channel 1 type
```

```
<< channel 1 type = cond28
```

Confirm the channel type.

```
>> calibration 1 datetime = 20171201000000, c0 = 0.2346, c1 = 153.4873, c2 = 1.0001
```

Set the core coefficients.

```
>> calibration 1 datetime = 20171201000000, x0 = 0.2003, x1 = 0.2943, x2 = 0.0051, x3 =
0.085, x4 = 0.0001, x5 = 0.0, x6 = 15.0280, x7 = 10.0025
```

Set the cross-channel correction coefficients.

```
>> calibration 1
<< calibration 1 label = conductivity_00, datetime = 20171201000000, c0 = 0.2346, c1 =
153.4873, c2 = 1.0001 x0 = 0.2003, x1 = 0.2943, x2 = 0.0051, x3 = 0.0850, x4 = 0.0001,
x5 = 0.0, x6 = 15.0280, x7 = 10.0025, n0 = 7, n1 = 3
```

Request confirmation of all calibration coefficients.

6.4 Supporting material

6.4.1 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. Fofonoff 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

$$S = a_0 + a_1 \cdot RT^{0.5} + a_2 \cdot RT^{1.0} + a_3 \cdot RT^{1.5} + a_4 \cdot RT^{2.0} + a_5 \cdot RT^{2.5} + \Delta S$$

where ΔS is a temperature correction term given by

$$\Delta S = \frac{T - 15}{1 + 0.0162 \cdot (T - 15)} \cdot Fn(RT)$$

where Fn(RT) is the polynomial function

$$Fn(RT) = b_0 + b_1 \cdot RT^{0.5} + b_2 \cdot RT^{1.0} + b3 \cdot RT^{1.5} + b_4 \cdot RT^{2.0} + b_5 \cdot RT^{2.5}$$

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.

$$RT = \frac{R}{Rp \cdot rT}$$

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 = 15°C, and P = 0: Conductivity(35, 15, 0) = 42.914 mS/cm.

$$R = \frac{Conductivity(S, T, P)}{Conductivity(35, 15, 0)}$$

Rp and rT are correction terms to adjust for in-situ pressure and temperature respectively.

$$Rp = 1 + \frac{e_1 \cdot P + e_2 \cdot P^2 + e_3 \cdot P^3}{1 + d_1 \cdot T + d_2 \cdot T^2 + (d_3 + d_4 \cdot T) \cdot R}$$

$$rT = c_0 + c_1 \cdot T + c_2 \cdot T^2 + c_3 \cdot T^3 + c_4 \cdot T^4$$

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-5
2	25.3851	-0.0066	1.104259e-4	4.464e-4	-6.370e-10
3	14.0941	-0.0375	-6.968e-7	0.4215	3.989e-15
4	-7.0261	0.0636	1.0031e-9	-3.107e-3	
5	2.7081	-0.0144			

7 Error messages

This is a current, but partial, list of error messages which the logger can produce when responding to issued commands. Each error 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. The following errors have been categorized in what is relevant to a misuse of the command set (wrong usage) and what is due to a hardware failure or a factory misconfiguration.
In case of a hardware failure, one course of action is to apply a full hardware reset (see Tips for system integrators).

7.1 List of error and warning messages

Error code	Reported description	Root cause
E0101	command parser busy	Wrong usage
E0102	invalid command '<*unknown-command-name*>'	Wrong usage
E0103	protected command, use 'permit command = <*command*>'	Wrong usage
E0104	feature not yet implemented	Wrong usage
E0105	command prohibited while logging	Wrong usage
E0107	expected argument missing	Wrong usage
E0108	invalid argument to command: '<*invalid-argument*>'	Wrong usage
E0109	feature not available	Wrong usage
E0110	buffer full	Wrong usage/Hardware failure
E0111	command failed	Hardware failure
E0114	feature not supported by hardware	Wrong usage
E0301	memory erase not completed	Hardware failure
E0401	estimated memory usage exceeds capacity	*This is a warning only*

Error code	Reported description	Root cause
E0402	memory not empty, erase first	Wrong usage
E0403	end time must be after start time	Wrong usage
E0404	end time must be after current time	Wrong usage
E0405	failed to enable for logging	Hardware failure
E0410	no sampling channels active	Wrong usage
E0411	period not valid for selected mode	Wrong usage
E0412	burst parameters inconsistent	Wrong usage
E0413	period too short for serial streaming	Wrong usage
E0414	thresholding interval not valid	Wrong usage
E0415	more than one gating condition is enabled	Wrong usage
E0416	wrong regimes settings	Wrong usage
E0417	no gating allowed with regimes mode	Wrong usage
E0418	cast detection needs a pressure/depth channel	Wrong usage
E0419	calibration coefficients are missing	Factory misconfiguration
E0420	required channel is turned off; <channel-index>	Wrong usage
E0421	raw output format not allowed	Wrong usage
E0422	AUX1 not available in current serial mode	Wrong usage
E0423	wrong ddsampling settings	Wrong usage
E0425	invalid settings	Wrong usage
E0426	postprocessing already active	Wrong usage
E0427	wrong memory format	Wrong usage
E0428	postprocessing reference channel not available	Wrong usage

Error code	Reported description	Root cause
E0501	item is not configured	Factory misconfiguration
E0505	no channels configured	Factory misconfiguration
E0601	no calibration for channel '<channel-index>'	Factory misconfiguration
"700" messages apply only if control of external devices is supported.		
E0701	device error: <*details*>	Hardware failure
E0702	no devices configured	Factory misconfiguration
E0703	device schedule inconsistent	Wrong usage
E0704	device is not enabled	Wrong usage
E0705	multiple operations not supported: '<extra-operation>'	Wrong usage