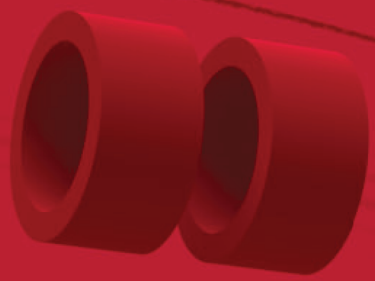
RBR

COMMAND REFERENCE

Generation 4

L3.5 version



rbr-global.com

Table of contents

1 Introduction

This document describes the Generation4 API (also referred to as Gen4 API). RBR instruments supporting the Gen4 API can be identified using the id or instrument command. Instruments supporting the Gen4 API include compact instruments (SL4), standalone sensors (SEN4), standard instruments (L4), and many OEM instruments.

1.1 Command processing and timeouts

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

```
<< ERR-101 command parser busy
```

1.1.1 Timeouts, output blanking, and power saving

1.1.1.1 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 fwversion=1.000 sn=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.
```

1.1.1.2 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.

> Data which would have been streamed but has been suppressed, due to output blanking, are dropped and so will never be streamed.

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.

1.1.1.3 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.

1.1.2 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.

1.1.3 Parameter naming constraints

When specifying labels for datasets, configurations, schedules or groups, there are a few constraints which must be respected:

- Maximum length of 31 characters.
- Must not start with a full stop/period character (`.`).
- Must not match any command name, critical parameter keywords, or existing labels; in this context a match is *not* case sensitive. A complete list of reserved words can be found in the appendix.
- Labels are case sensitive; '**a**' is not the same as '**A**'.
- It is good practice to use only alphabetic characters, numeric digits, and the underscore '_' character. This will avoid problems in future if any other punctuation character is assigned a special purpose.

Special characters which can not be used:

Extended ASCII characters (code 128+)

' ' (space)

'!' (exclamation mark)

'“' (double quote)

'#' (number sign)

'$' (dollar)

'&' (ampersand)

''' (single quote)

‘()' (parentheses)

'*' (asterisk)

',' (comma)

'/' (forward slash)

';' (semicolon)

'<' (less than)

'='(equals)

'>' (greater than)

'?' (question mark)

'@' (at sign)

'[]' (square brackets)

'\' (backslash)

'^' (caret)

'` ' (grave)

'{}' (curly brackets)

'|' (pipe)

'~' (tilde)

'%' (percent)

1.1.4 Command entry

1.1.4.1 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:

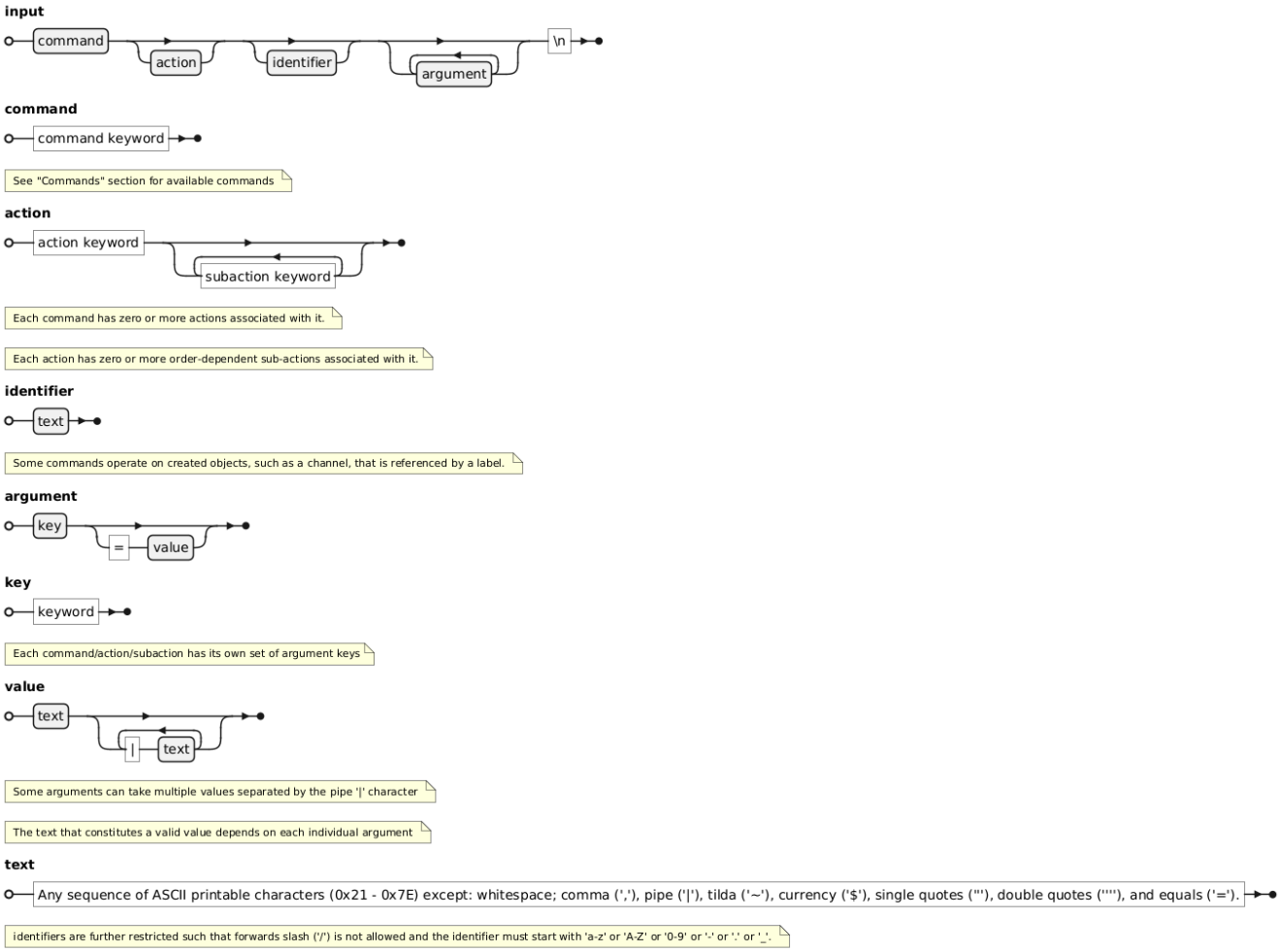
```
>> <CR><LF>
<< ready:

>> <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 **settings prompt** command).

1.1.4.2 Case sensitivity

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.

1.1.4.3 Grammar



1.1.4.4 Common error messages

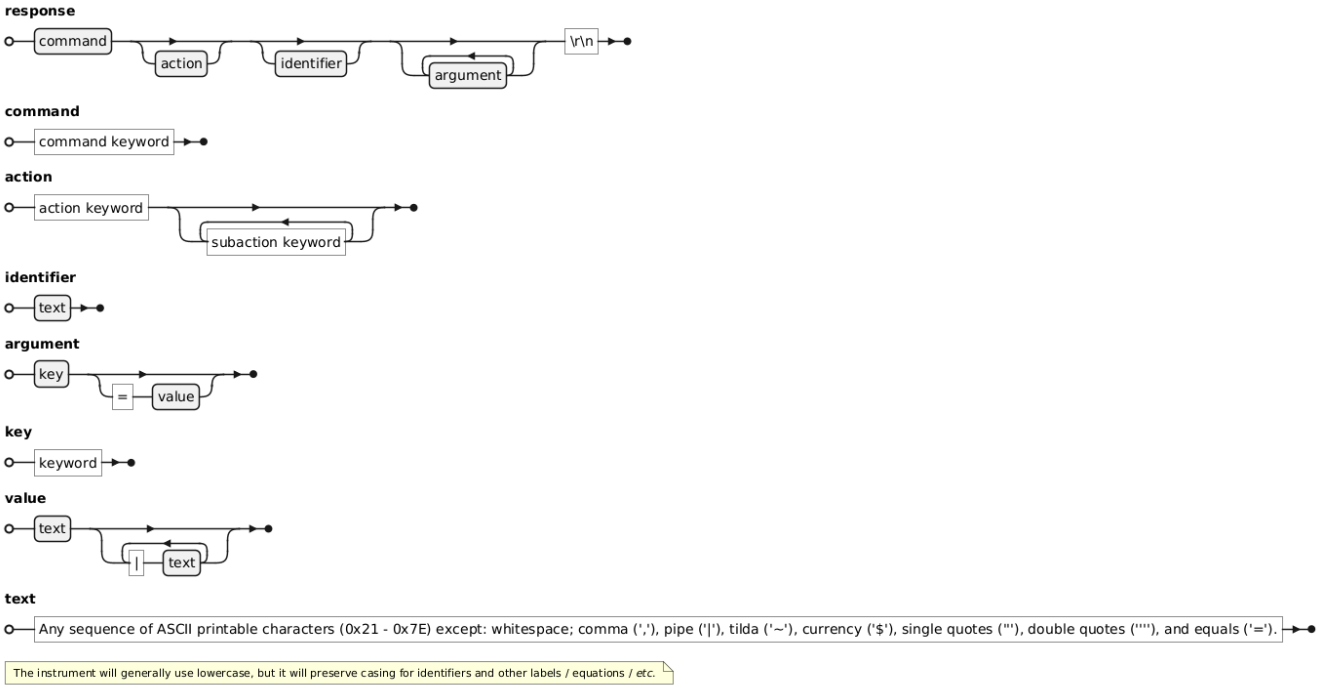
The received message may or may not form a valid command; errors detectable by the logger will vary from one command to another, but some of the common, general errors include:

- ERR-102 invalid command '<unknown_text>'
- ERR-107 expected argument missing
- ERR-108 invalid argument to command: '<unknown_text>'
- See Information, warning, and error codes for a complete list.

1.1.5 Parsing logger responses

1.1.5.1 Responses to commands

Responses to commands follow the grammar described below.



There are some important points to consider when implementing robust automated parsing of responses to logger commands:

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

  ```
  >> STORAGE USED
  << storage 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,

  ```
  >> storage size used remaining
  << storage 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 undocumented in this version of the Gen4 command reference. The reporting order is also subject to change from one firmware version to another. For example,

  ```
  >> storage
  << storage used=0 remaining=132120576 size=132120576
  ```

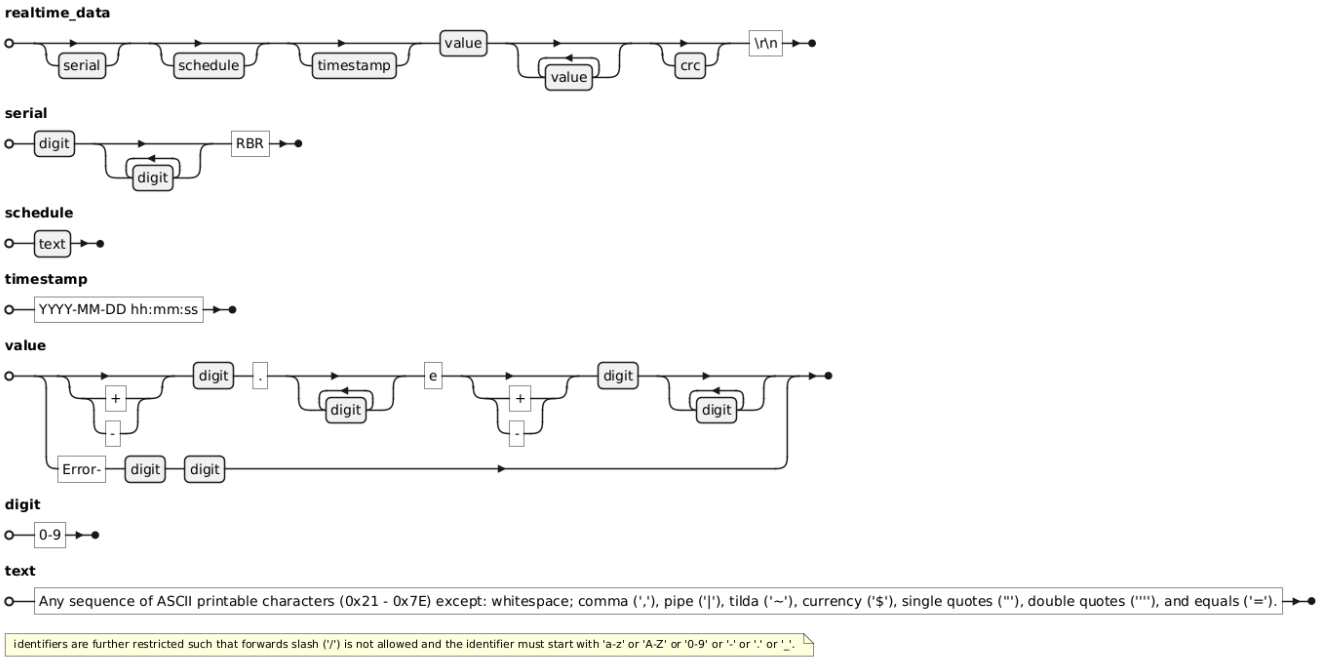
  is the current behaviour, but a future version might respond as follows:

```
>> storage
<< storage 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.

1.1.5.2 Parsing streamed or polled samples

Streamed or polled samples follow the grammar described below.



There are some important points to consider when implementing robust parsing of streamed or polled samples:

- When parsing streamed output or polled values, it is good practice to assume that any channel could report an error code (see outputformat and poll commands). If a channel reports an error code, other channels might still be valid.
- When handling realtime data, depending on group and schedule composition, relatively large amounts of data can be produced since channels will be repeated for every group membership.
  An estimate of the maximum expected size can be calculated by the formula: bytes = 70 + 24 x number of channels in the schedule.

  So, if an instrument has “channel1”, “channel2”, “channel3”, “channel4”; with groups: “group1 = channel2|channel1”, “group2 = “channel1|channel2|channel4”; and schedule: “schedule1 = group1|group2”, which gives five channels in the schedule; the output for realtime would be akin to:

```
<< RBR 123456 schedule1 2025-01-01 00:00:00:00 channel2_value channel1_value
channel1_value channel2_value channel4_value CRC
```

1.1.5.3 Parsing numbers

- 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 turbidity_00 gain
<< channel turbidity_00 gain=auto

>> channel turbidity_00 gain=20
<< channel turbidity_00 gain=20.0
```

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

- 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, see Sample data storage 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.

1.2 Formatting

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

Access to some instrument commands is restricted or controlled in certain situations. These controls are referred to as Security settings, and there are currently two:

- **Open** commands can be executed without restriction.
- **Unsafe** commands are those which the logger will not execute if logging is in progress. For example, a sampling period cannot be changed in the middle of a deployment. Reading parameters is always available.

Some commands may allow interrogation while the instrument is enabled (open) but modification must be done when the instrument is disabled (unsafe).

2 Quick start

This section details commands sent to (**>>**) and responses received from (**<<**) the instrument 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 instrument.

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

2.1 General overview

2.1.1 Acquiring samples

There are basically two fundamental ways to acquire samples: they can be acquired by polling or according to a defined sampling schedule.

A polled measurement is performed simply via the poll command. A sample acquired *only* for polling purposes is not stored in the logger's memory.

Scheduled samples are configured using various commands. Scheduled samples may be stored in the logger's memory, and they can also be streamed directly out of the logger in real-time; see the stream and storage parameters of the schedule command. Gen-4 instruments have the ability to sample with different measurement parameters according to different schedules, and for this, we need to understand the concepts of **channels**, **groups**, **schedules** and **configurations**. These will be described briefly in the sections below, but briefly:

- Channels can be arranged in groups.
- Each group may be associated with one or more schedules.
- Each schedule can have its own sampling mode and parameters, independent of other schedules.
- A sampling mode can be as simple as a single fixed sampling rate, or as complex as a multi-rate scheme driven by the logger's depth in the water.
- One or more schedules are collected together into a configuration.
- The logger is enabled using one of these configurations, and it will execute the associated schedule(s) to acquire data.

Gen-4 instruments are also designed to support retention of data in the memory from more than one deployment; it will no longer be necessary to delete the previous deployment's data before starting a new one.

A major feature of RBR instruments is that polled and scheduled samples can occur at the same time, they are not mutually exclusive.

The acquisition of data can be constrained or controlled in other ways, by configuring a gating condition such as time and twist activation. If available, these features apply to the entire instrument, not on a per-schedule basis.

Refer to the remaining Quick start sections for examples of different sequences of commands used to set up different ways to acquire samples.

2.1.2 Channels

Channels are the basic elements of what will become a set of data acquired by the instrument. As with previous generations of RBR instruments, Gen-4 products can measure a wide range of physical properties of the water, such as temperature, pressure, conductivity, turbidity, dissolved oxygen, chlorophyll, and so on. This list is not exhaustive and the number of sensors supported continues to increase regularly. For each one of these physical properties measured - for example pressure - the instrument has exactly one channel. Instrument channels may also be available for physical properties that are not measured directly, but that are derived by calculation from measured channels - for example salinity, which is derived from conductivity, temperature and pressure. In summary:

- A channel represents a single parameter of interest recorded by the logger. The parameter may be directly measured or derived by calculation from other parameters.
- Although channels themselves are not a new concept in RBR instruments, the way in which they are specified for sampling is new for Gen-4.
- Users can not create or delete channels. Modification of channel details is limited to its calibration or other specialised parameters (for example gain, if applicable).
- Channels are created and assigned a label at the factory when the logger is built.
- Any channel can be a member of more than one **group**.
- One **configuration** is selected when the logger is enabled; this specifies, via the list of **schedules**, which groups will be sampled. Any channel not belonging to at least one of these groups will not be sampled during the deployment.

When sending commands that access or modify information associated with a channel, the channel is referred to by its label - for example, `temperature_00`.

Associated command(s):

channel

calibration

sensor

2.1.3 Groups

- A group is a collection of one or more logger channels.
- In the context of sampling, a group (or a list of groups) determines which channels will be sampled according to a given schedule.
- Channels can not be directly assigned to a schedule; this can only be done through a group. A group may contain only one channel if necessary.
- The logger can maintain a pool of groups, any of which can be included in the group list of one or more schedules.
- The user has complete control over the creation, content, and deletion of groups.
- When creating a group, the user assigns it a label; group labels must be unique.
- One configuration is selected when the logger is enabled; any group that is not associated with a schedule included in that configuration will not be sampled as part of the deployment.

Associated command(s):

group

2.1.4 Schedules

- A schedule determines how a given subset of groups will be sampled during the deployment.
- Multiple schedules can be included in a configuration, enabling groups to be sampled in different ways.
- The logger can maintain a pool of schedules, any of which can be included in one or more configurations.
- The user has complete control over the creation, content, and deletion of schedules.
- Each schedule contains a list of channel groups, one sampling mode, and a set of sampling parameters appropriate for the mode.
- When creating a schedule, the user assigns it a label; schedule labels must be unique.
- One configuration is selected when the logger is enabled; only the schedules included in that configuration will be executed during the deployment.

Associated command(s):

schedule

2.1.5 Configurations

- A configuration is essentially a list of the sampling schedules to be executed by the logger when it is enabled.
- The logger can maintain a pool of configurations; when the logger is enabled, the user specifies exactly one configuration that determines how the logger will behave during the deployment.
- The user has complete control over the creation, content, and deletion of configurations; if necessary, the logger can always be restored to its as-shipped factory default configuration.
- When creating a configuration, the user assigns it a label; configuration labels must be unique.
- All elements of a configuration must be valid and consistent before it can be used to enable the logger.
- The selected configuration forms a subset of the metadata for the deployment, and a copy is stored in memory when the logger is enabled.

Associated command(s):

config

2.1.6 Datasets

- A dataset is the collection of all sample data and all supporting auxiliary data relating to a single deployment of the logger, stored in the logger's data memory.
- It comprises a number of storage objects; for convenience, these storage objects can be thought of as "files", and will be referred to as such. However, they may not necessarily be implemented as actual files within the memory.
- One file contains the deployment metadata; this is a snapshot of the logger's state and parameters at the time the deployment was enabled.
- A second file contains all the events that occurred during the deployment. Essentially, an event is a record of anything that occurs during the deployment that is not sample data.
- The remaining file(s) contain the sample data, one file for each schedule defined for the deployment.
- The user creates a dataset by enabling the logger using the enable command, assigning it a label and associating it with a configuration.
- Only one dataset can be active (reflected by the status 'open', see **dataset** command). As the deployment progresses, sample data and events accumulate within the active dataset.
- If logging is stopped (for any reason), the active dataset becomes one of a number of historical datasets retained in memory, and will not be updated any further (reflected by the status 'closed', see **dataset** command).
- The user can delete any historical dataset to increase the amount of memory available for new ones.

Associated command(s):

dataset

enable

disable

download

2.2 Enabling continuous sampling

Start sampling immediately using the default configuration built into the instrument.

Assuming the default configuration is as:

```
>> group create gr_pts
<< group create gr_pts
>> group gr_pts channellist=pressure_00|salinity_00|temperature_00
<< group gr_pts channellist=pressure_00|salinity_00|temperature_00

>> schedule create sch_asc_pts
<< schedule create sch_asc_pts
>> schedule sch_asc_pts grouplist=gr_pts mode=continuous
<< schedule sch_asc_pts grouplist=gr_pts mode=continuous

>> schedule sch_asc_pts grouplist=gr_pts stream=off storage=on mode=continuous
period=1000 castdetection=off
<< schedule sch_asc_pts grouplist=gr_pts stream=off storage=on mode=continuous
period=1000 castdetection=off

>> config create default_config
<< config create default_config
>> config default_config schedulelist=sch_asc_pts
<< config default_config schedulelist=sch_asc_pts
```

Ensure no time gating on deployment:

```
>> deployment gate=none
<< deployment gate=none
```

Verify deployment:

```
>> verify config=default_config dataset=dataset_01
<< verify config=default_config dataset=dataset_01 storagemode=normal state=enabled
```

Enable deployment:

```
>> enable config=default_config dataset=dataset_01
<< enable config=default_config dataset=dataset_01 storagemode=normal state=enabled
```

Check current deployment status:

```
>> deployment status
<< deployment status=sampling
```

2.3 Serial streaming from serial port

2.3.1 Setting the correct baud rate

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

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

>> link serial baudrate=115200
<< link serial baudrate=115200
% This response is sent at the old baudrate, 19200Bd.
% The host must now change its baudrate to 115200Bd.
```

2.3.2 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 schedule command). If the instrument memory is full, the instrument will continue streaming as long as the instrument should be logging (see deployment command). The **instrument outputformat** command indicates which channels will be reported and sets the type of output format to be used.

With an RBRconcerto^3 C.T.D, assuming the configuration is set as:

```
>> group create gr_pts
<< group create gr_pts
>> group gr_pts channellist=pressure_00|temperature_00|salinity_00
<< group gr_pts channellist=pressure_00|temperature_00|salinity_00

>> schedule create sch_asc_pts
<< schedule create sch_asc_pts
>> schedule sch_asc_pts grouplist=gr_pts mode=continuous period=1000
<< schedule sch_asc_pts grouplist=gr_pts mode=continuous period=1000

>> config create default_config
<< config create default_config
>> config default_config schedulelist=sch_asc_pts
<< config default_config schedulelist=sch_asc_pts
```

Ensuring the schedule is streamed out with the right format:

```
>> schedule sch_asc_pts stream=serial
<< schedule sch_asc_pts stream=serial
```

Set output format:

```
>> instrument outputformat sn=on schedulelabel=on crc=on encoding=ascii
<< instrument outputformat sn=on schedulelabel=on crc=on encoding=ascii
```

Enabling the instrument:

```
>> enable config=default_config dataset=dataset_01
<< enable config=default_config dataset=dataset_01 storagemode=normal state=enabled
```

Instrument starts streaming measurements:

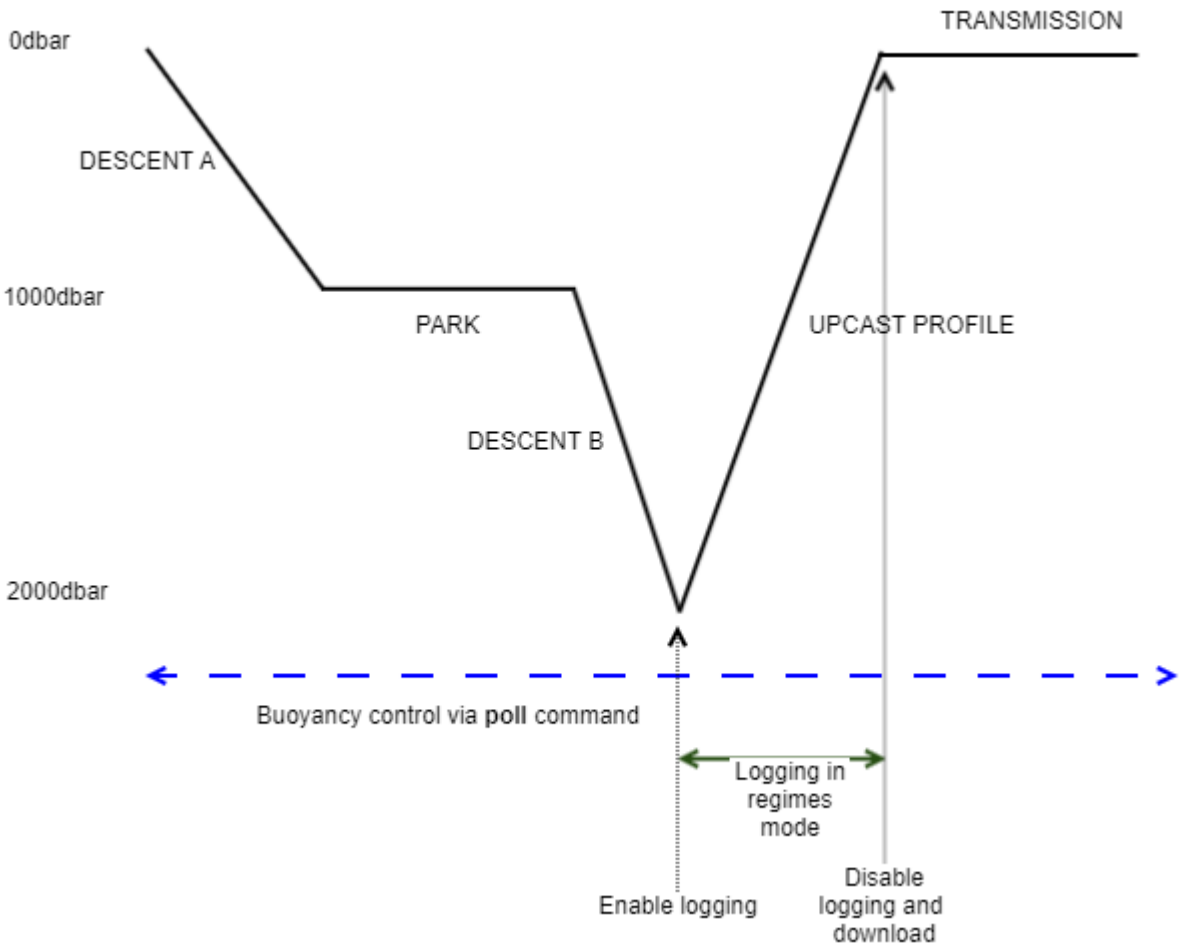
```
<< RBR 999999 sch_asc_pts 2024-01-01 00:10:14.000 10.0110e+000 21.3213e+000 0x94AB
<< RBR 999999 sch_asc_pts 2024-01-01 00:10:15.000 10.0241e+000 21.0201e+000 0x3A3A
<< RBR 999999 sch_asc_pts 2024-01-01 00:10:16.000 10.0248e+000 21.8246e+000 0x5967
```

2.4 Integrating with a profiling float

2.4.1 Introduction

There are a number of dedicated features in the RBR*argo* 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.



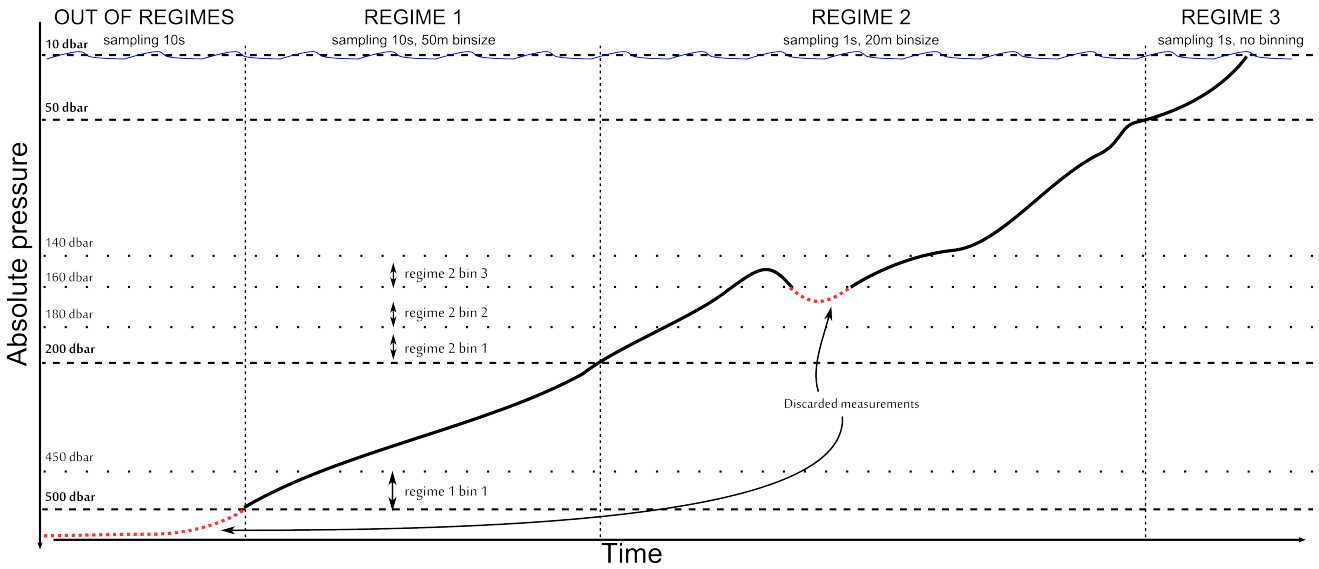
2.4.2 Buoyancy control

When operated on a float, the RBR instrument is used mostly as a depth sensor, providing input to the buoyancy engine and float controller. This is typically done interactively using the poll 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 **poll channellist=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 poll command by following the recommended wake-up procedure.

2.4.3 Regime sampling mode

The real science of the Argo profiler occurs during the upcast of the float, 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 for a 500dbar profile.



As shown above, three distinct sampling regimes are required. Each 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 internal measurement rate (msec).
- The bin size dictates the amount of water column (in dbar) over which the samples will be averaged and stored.

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

The following chapters are various examples of how to operate the RBRargo.

2.4.4 Single schedule, single configuration RBR*argo* C.T.D example

The following is a concrete example of how to operate an RBR*argo* C.T.D using the regime sampling mode and how to enable/disable/download the data.

> Some systems rely on instruments streaming data rather than storing it. The stream and storage parameters of the **schedule** command can accomplish this.

The RBR*argo* C.T.D is configured to collect measurements during the ascent phase:

- Salinity, temperature, and absolute pressure sampled
  - 0.1Hz binned 50dbar between 500dbar and 200dbar
  - 1Hz binned 20dbar between 200dbar and 50dbar
  - 1Hz non-binned from 50dbar to the surface

2.4.4.1 Beginning of mission, initial configuration

2.4.4.1.1 Ensuring default state

```
>> disable
<< disable state=disabled

>> dataset delete all
<< dataset delete all

>> config delete all
<< config delete all

>> schedule delete all
<< schedule delete all

>> group delete all
<< group delete all
```

2.4.4.1.2 Group definition

In this example, only an RBR*argo*^{3} C.T.D is available on the float. The float will acquire and transmit the pressure, the salinity (with dynamic correction) and the temperature:

```
>> group create gr_pts
<< group create gr_pts
>> group gr_pts channellist=seapressure_00|salinitydyncorr_00|temperature_00
<< group gr_pts channellist=seapressure_00|salinitydyncorr_00|temperature_00
```

2.4.4.1.3 Schedule definition

The following schedule definition follows the example above (see diagram).

```
>> schedule create sch_asc_pts
<< schedule create sch_asc_pts
>> schedule sch_asc_pts grouplist=gr_pts stream=off storage=on mode=regimes
<< schedule sch_asc_pts grouplist=gr_pts stream=off storage=on mode=regimes
```

```
>> schedule sch_asc_pts direction=ascending count=3 reference=seapressure_00
finalboundary=0 boundary1=500 binsize1=50 period1=10000 boundary2=200
binsize2=20 period2=1000 boundary3=50 binsize3=0 period3=1000
<< schedule sch_asc_pts direction=ascending count=3 reference=seapressure_00
finalboundary=0 boundary1=500 binsize1=50.0 period1=10000 boundary2=200
binsize2=20.0 period2=1000 boundary3=50 binsize3=0.0 period3=1000
```

2.4.4.1.4 Configuration definition

```
>> config create cf_ascent
<< config create cf_ascent
>> config cf_ascent schedulelist=sch_asc_pts
<< config cf_ascent schedulelist=sch_asc_pts
```

2.4.4.1.5 Deployment parameters

Ensure the deployment gate condition is set to none to ensure sample acquisition is entirely based on the sampling mode:

```
>> deployment gate=none
<< deployment gate=none
```

2.4.4.2 Start of ascent

Ensures the memory is cleared first:

```
>> dataset delete all
<< dataset delete all
```

Enable the instrument:

```
>> enable config=cf_ascent dataset=ds_ascent
<< enable config=cf_ascent dataset=ds_ascent storagemode=normal state=enabled
```

2.4.4.3 End of ascent, disable logging and download data

Downloading the data from the instrument can be done while a schedule is still enabled (i.e. when the float is ascending), 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 state=disabled
```

Determine how much memory has been used:

```
>> dataset ds_ascent/sch_asc_pts/data bytecount
<< dataset ds_ascent/sch_asc_pts/data bytecount=3501260
```

Now loop over the data to download it in chunks:

```
>> download ds_ascent/sch_asc_pts/data bytecount=<chunk-size> bytestart=0
<< download ds_ascent/sch_asc_pts/data bytecount=<chunk-size> bytestart=0<cr><lf>
<bytes[start…bytecount]-of-data><crc>

>> download
<< download ds_ascent/sch_asc_pts/data bytecount=<chunk-size> bytestart=<1 × chunk-
size><cr><lf> <bytes[start…bytecount]-of-data><crc>

>> download
<< download ds_ascent/sch_asc_pts/data bytecount=<chunk-size> bytestart=<2 × chunk-
size><cr><lf> <bytes[start…bytecount]-of-data><crc>
...
...
>> download
<< download ds_ascent/sch_asc_pts/data bytecount=<final-chunk-size> bytestart= <(n - 1)
× chunk-size><cr><lf> <bytes[start…bytecount]-of-data><crc>
```

The data returned by the download 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 Sample data storage 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.

2.4.5 RBRargo BGC with multiple schedules and different configurations

In the context of a BGC Argo float with many sensors, having different schedules for each sensor helps save power and can dramatically increase the lifetime of the float. This is easily achieved with the instrument by configuring multiple groups/schedules.

Here is a concrete example of an Argo BGC float configuration.

During ascending:

- Salinity, temperature, and pressure sampled
  - 1Hz binned 10dbar between 2000dbar and 1000dbar
  - 1Hz binned 1dbar between 1000dbar and 50dbar
  - 1Hz non-binned between 50dbar and 0dbar
- ODO concentration, ODO temperature sampled
  - 20s binned 10dbar between 1000dbar and 250dbar
  - 5s binned 2dbar between 250dbar and 0dbar
- pH
  - 500s non-binned between 1000dbar and 250dbar
  - 20s non-binned 250dbar and 0dbar
- BBP700/Chlorophyll/FDOM
  - 10s binned 10dbar between 2000dbar and 250dbar
  - 5s binned 2dbar between 250dbar and 0dbar
- Downwelling PAR and irradiance at 412nm, at 443nm, at 490nm
  - 5s binned 2dbar between 250dbar and 0dbar

During park:

- Salinity, temperature, pressure
  - continuous every 6h
- ODO concentration, ODO temperature
  - continuous every 12h

⚠ When using one channel on one schedule and the same channel (or one of its supporting channels) on another schedule, their sampling frequency should be multiple. See schedule command for more details.

2.4.5.1 Beginning of the mission, initial configuration

2.4.5.1.1 Ensuring default state

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

>> dataset delete all
<< dataset delete all

>> config delete all
<< config delete all

>> schedule delete all
<< schedule delete all

>> group delete all
<< group delete all
```

2.4.5.1.2 Group definitions

PTS group

```
>> group create gr_pts
<< group create gr_pts
>> group gr_pts channellist=seapressure_00|salinitydyncorr_00|temperature_00
<< group gr_pts channellist=seapressure_00|salinitydyncorr_00|temperature_00
```

ODO group

```
>> group create gr_odo
<< group create gr_odo
>> group gr_odo channellist=seapressure_00|oxygenconcentration_00|odotemperature_00
<< group gr_odo channellist=seapressure_00|oxygenconcentration_00|odotemperature_00
```

pH group

```
>> group create gr_ph
<< group create gr_ph
>> group gr_ph channellist=seapressure_00|ph_00
<< group gr_ph channellist=seapressure_00|ph_00
```

BBPFL group

```
>> group create gr_bbpfl
<< group create gr_bbpfl
>> group gr_bbpfl channellist=seapressure_00|backscatter_00|chlorophyll_00|fdom_00
<< group gr_bbpfl channellist=seapressure_00|backscatter_00|chlorophyll_00|fdom_00
```

Radiometry group

```
>> group create gr_radiometry
<< group create gr_radiometry
>> group gr_radiometry channellist=seapressure_00|par_00|irradiance_00|irradiance_01|
irradiance_02
<< group gr_radiometry channellist=seapressure_00|par_00|irradiance_00|irradiance_01|
irradiance_02
```

2.4.5.1.3 Schedule definitions

PTS ascending schedule

```
>> schedule create sch_asc_pts
<< schedule create sch_asc_pts
>> schedule sch_asc_pts grouplist=gr_pts stream=off storage=on mode=regimes
<< schedule sch_asc_pts grouplist=gr_pts stream=off storage=on mode=regimes
>> schedule sch_asc_pts direction=ascending count=3 reference=seapressure_00
finalboundary=0 boundary1=2000 binsize1=10 period1=1000 boundary2=1000
binsize2=1 period2=1000 boundary3=50 binsize3=0 period3=1000
<< schedule sch_asc_pts direction=ascending count=3 reference=seapressure_00
finalboundary=0 boundary1=2000 binsize1=10.0 period1=1000 boundary2=1000
binsize2=1.0 period2=1000 boundary3=50 binsize3=0.0 period3=1000
```

ODO ascending schedule

```
>> schedule create sch_asc_odo
<< schedule create sch_asc_odo
>> schedule sch_asc_odo grouplist=gr_odo stream=off storage=on mode=regimes
<< schedule sch_asc_odo grouplist=gr_odo stream=off storage=on mode=regimes
>> schedule sch_asc_odo direction=ascending count=2 reference=seapressure_00
finalboundary=0 boundary1=1000 binsize1=10 period1=20000 boundary2=250
binsize2=2 period2=5000
<< schedule sch_asc_odo direction=ascending count=2 reference=seapressure_00
finalboundary=0 boundary1=1000 binsize1=10.0 period1=20000 boundary2=250
binsize2=2.0 period2=5000
```

pH ascending schedule

```
>> schedule create sch_asc_ph
<< schedule create sch_asc_ph
>> schedule sch_asc_ph grouplist=gr_ph stream=off storage=on mode=regimes
<< schedule sch_asc_ph grouplist=gr_ph stream=off storage=on mode=regimes
```

```
>> schedule sch_asc_ph direction=ascending count=2 reference=seapressure_00
finalboundary=0 boundary1=1000 binsize1=0 period1=500000 boundary2=250 binsize2=0
period2=20000
<< schedule sch_asc_ph direction=ascending count=2 reference=seapressure_00
finalboundary=0 boundary1=1000 binsize1=0.0 period1=500000 boundary2=250 binsize2=0.0
period2=20000
```

BBPFL ascending schedule

```
>> schedule create sch_asc_bbpfl
<< schedule create sch_asc_bbpfl
>> schedule sch_asc_bbpfl grouplist=gr_bbpfl stream=off storage=on mode=regimes
<< schedule sch_asc_bbpfl grouplist=gr_bbpfl stream=off storage=on mode=regimes
>> schedule sch_asc_bbpfl direction=ascending count=2 reference=seapressure_00
finalboundary=0 boundary1=2000 binsize1=10 period1=10000 boundary2=250 binsize2=2
period2=5000
<< schedule sch_asc_bbpfl direction=ascending count=2 reference=seapressure_00
finalboundary=0 boundary1=2000 binsize1=10.0 period1=10000 boundary2=250 binsize2=2.0
period2=5000
```

Radiometry ascending schedule

```
>> schedule create sch_asc_radiometry
<< schedule create sch_asc_radiometry
>> schedule sch_asc_radiometry grouplist=gr_radiometry stream=off storage=on
mode=regimes
<< schedule sch_asc_radiometry grouplist=gr_radiometry stream=off storage=on
mode=regimes
>> schedule sch_asc_radiometry direction=ascending count=1 reference=seapressure_00
finalboundary=0 boundary1=250 binsize1=2 period1=5000
<< schedule sch_asc_radiometry direction=ascending count=1 reference=seapressure_00
finalboundary=0 boundary1=250 binsize1=2.0 period1=5000
```

PTS park schedule

```
>> schedule create sch_park_pts
<< schedule create sch_park_pts
>> schedule sch_park_pts grouplist=gr_pts stream=off storage=on mode=continuous
period=21600000 castdetection=off
<< schedule sch_park_pts grouplist=gr_pts stream=off storage=on mode=continuous
period=21600000 castdetection=off
```

ODO park schedule

```
>> schedule create sch_park_odo
<< schedule create sch_park_odo
>> schedule sch_park_odo grouplist=gr_odo stream=off storage=on mode=continuous
period=43200000 castdetection=off
<< schedule sch_park_odo grouplist=gr_odo stream=off storage=on mode=continuous
period=43200000 castdetection=off
```

2.4.5.1.4 Configuration definitions

Ascent configuration

```
>> config create cf_ascent
<< config create cf_ascent
>> config cf_ascent schedulelist=sch_asc_pts|sch_asc_odo|sch_asc_ph|sch_asc_bbpfl|
sch_asc_radiometry
<< config cf_ascent schedulelist=sch_asc_pts|sch_asc_odo|sch_asc_ph|sch_asc_bbpfl|
sch_asc_radiometry
```

Park configuration

```
>> config create cf_park
<< config create cf_park
>> config cf_park schedulelist=sch_park_pts|sch_park_odo
<< config cf_park schedulelist=sch_park_pts|sch_park_odo
```

2.4.5.1.5 Deployment parameters

Ensure the deployment gate condition is set to none to ensure sample acquisition is entirely based on the sampling mode:

```
>> deployment gate=none
<< deployment gate=none
```

2.4.5.2 Ascent mode

Just before ascent, enable the instrument in the ascent configuration:

```
>> enable config=cf_ascent dataset=ds_ascent
<< enable config=cf_ascent dataset=ds_ascent storagemode=normal state=enabled
```

2.4.5.3 End of ascent

Disable the ongoing deployment:

```
>> disable
<< disable state=disabled
```

Read info about available datasets:

```
<< dataset list
>> dataset list=ds_ascent

>> dataset ds_ascent status schedulelist
<< dataset ds_ascent status=closed schedulelist=sch_asc_pts|sch_asc_odo|sch_asc_ph|
sch_asc_bbpfl|sch_asc_radiometry
```

Get data bytecount for each schedule. Take the first schedule as an example:

```
<< dataset ds_ascent/sch_asc_pts/data
>> dataset ds_ascent/sch_asc_pts/data bytecount=123456 samplecount=6172 datatype=float32
```

Download the PTS data by looping over the data to download it in chunks:

```
>> download ds_ascent/sch_asc_pts/data bytecount=1024 bytestart=0
<< download ds_ascent/sch_asc_pts/data bytecount=1024
bytestart=0<cr><lf><1024bytes><crc>

<< download
>> download ds_ascent/sch_asc_pts/data bytecount=1024
bytestart=1024<cr><lf><1024bytes><crc>
....
<< download
>> download ds_ascent/sch_asc_pts/data bytecount=560
bytestart=122880<cr><lf><560bytes><crc>
```

Download the ODO data by looping over the data to download it in chunks:

```
>> download ds_ascent/sch_asc_odo/data bytecount=1024 bytestart=0
<< download ds_ascent/sch_asc_odo/data bytecount=1024
bytestart=0<cr><lf><1024bytes><crc>

<< download
>> download ds_ascent/sch_asc_odo/data bytecount=1024
bytestart=1024<cr><lf><1024bytes><crc>
....
<< download
>> download ds_ascent/sch_asc_odo/data bytecount=420
bytestart=6144<cr><lf><420bytes><crc>
```

Download pH, BBPFL, and radiometry data in the same way.

2.4.6 RBR*argo* introspection

The list of channels populated onto an instrument and used 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) and the RBR*argo* model (e.g. an RBR*argo* C.T.D vs an RBR*argo* C.T.D.ODO).

The following list is the available channels populated for an RBR*argo* C.T.D.

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)

Channel label	Description
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)
salinitydyncorr_00	Salinity with dynamic correction applied (PSU)
cnt_00	Counts, number of sample used to calculate a bin average, when not binning, this is reporting the value 1

A complete list of possible labels is provided in the section Channel labels.

For platforms handling different models of the RBRargo, it is possible to determine the available channels by just issuing the channels command:

```
>> channel list
<< channel list=conductivity_00|temperature_00|pressure_00|sea_pressure_00|salinity_00|
conductivitycelltemperature_00|temperaturedyncorr_00|salinitydyncorr_00|cnt_00
```

2.4.7 Providing platform details to end-users

Some end-users want to keep the history of the sensor's details (for example, the 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. However, some end users might want to have all possible information readily available from the log files sent by the float to the shore.
The **instrument dump**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 instrument commands in the log files.

2.4.8 Sensor drift monitoring at the 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:

```
>> calibration pressure_00 datetime=20240501140000 offset=0.2
<< calibration pressure_00 datetime=20240501140000 offset=200.00000e-3
```

Optical dissolved oxygen measurements in the air 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 the 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 waves might wash the conductivity cell. If only one conductivity value is to be transmitted to shore, always use the lowest value. If possible, it is preferable to report directly the conductivity and not the salinity as the PSS78 calculation will saturate at 0 (see pss78 - derivation of Practical Salinity (1978)).

2.4.9 Energy tracking

The instrument tracks energy consumed via the command **instrument power external**.

2.5 Tips for system integrators

2.5.1 Deployment start time

The command *deployment* allows the start time of a deployment to be accessed, modified, or ignored. This concept of a start time is beneficial mainly to standard product 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).

The use of the start time is dictated by the gating condition set. The ***time*** gating condition is the only one which will use the starttime to gate a deployment.

> 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.

If a deployment is required to start at a specific time in the future, configure the deployment gate for time as follows:

```
>> deployment starttime=20000101000000 gate=time
```

However, if the deployment is not required to start at a certain time then we recommend using ***none*** as the gating option:

```
>> deployment gate=none
```

2.5.2 Sampling rates

The schedule 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.

2.5.2.1 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 schedule command, is calculated as (using integer division): $T_s = \frac{1000 + \frac{F_s}{2}}{F_s}$

2.5.2.2 Converting from milliseconds to Hz

If Ts is the number of milliseconds reported by the logger as the period (see schedule command), then the corresponding sampling rate Fs in Hz, is calculated as (using integer division): $F_s = \frac{1000 + \frac{T_s}{2}}{T_s}$

2.5.2.3 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.

2.5.3 Future proofing development

The chapter Command Processing and Timeouts gives a good overview on best practices on how to parse and handles logger commands.

For OEM customers using more than one configuration or planning to, it is best practice to inspect which channels are available with the channel command:

```
>> channel
<< channel count=5 list=conductivity_00|temperature_00|pressure_00|sea_pressure_00|
salinity_00
```

2.5.4 Power management and power cycling behaviour

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

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

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 (see Note). In such cases, the date and time may be reset to 2000-01-01 00:00:00. However, if the clock is lost while logging, the instrument will try to reset the clock to midnight of the day following the last known good sample. For example, if the last sample stored was at 2024-03-17 15:34:26.000, the instrument would try to reset the clock to 2024-03-18 00:00:00.000 before continuing to log data. Although the instrument no longer has any true idea of what the actual date and time are, this strategy does at least ensure that stored (and streamed) timestamps continue to advance monotonically.

⚠ 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 might not be processed correctly.

If an instrument is power cycled while it is logging or streaming, it will continue logging and/or streaming, even if the RTC clock is reset as described above. This activity will continue after the initialization period, the settling time and the read time periods have expired (see channel) following the restoration of power.

In order to enforce a full hardware reset, this 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

ℹ The amount of time required for the onboard clock to lose the date and time following the removal of all power may vary between instrument types. For some it may be only a few minutes, for others it can be several hours.

2.5.5 Error handling

2.5.5.1 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 115200 bps (default baudrate if configured baudrate lost).

2.5.5.2 Instrument reporting a hardware failure

If the instrument is reporting a hardware failure error code as described in Information, warning, and error codes, one course of action is to apply a full hardware reset (as described above).

2.5.5.3 Instrument reporting error codes in the measurements

Most of the error codes reported 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 can be resolved 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 **poll** for 10 seconds, and enable the unit again.
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 resets will do.

In any case, only channels reported as Error should be discarded. A classic 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 but measurements from other channels are still valid and useful.

2.5.6 Electronic static discharge

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

2.6 Migrate from Gen3 to Gen4 platforms

2.6.1 Introduction

For customers who have used Gen3 API, migrating to the Gen4 API will require 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.

2.6.2 Identifying a Gen4 platform

Identifying a platform running the Gen4 API can be easily identified via the **fwtype** parameter of the **id** command.

In the case of the L3.5 platform (Gen4 API), one would obtain:

```
>> id fwtype
<< id fwtype=120
```

Whereas, in the case of the L3 platform (Gen3 API), one would obtain:

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

2.6.3 Removed commands

The following commands have been removed:

- **bsl**: replaced by the **instrument flash** command
- **channels**: replaced by the channel command
- **confirmation**: replaced by the settings command
- **ddsampling**: replaced by the schedule command
- **fetch**: replaced by the poll command.
  With L3 platform:

  ```
  >> fetch
  << 2023-01-04 11:50:49.000, 12.7052 dbar, 18.1745 C
  ```

  With L4 platform:

  ```
  >> poll
  << 2024-10-21 11:50:49.000 18.1745130 12.7052970 2.69308210
  ( The exact form of the output will also depend on the settings of the
  outputformat command. )
  ```

- **getall**: replaced by the **instrument dump** command
- **hwrev**: replaced by the pcba command
- **info**: replaced by the instrument command
- **memclear**: replaced by the **dataset delete** command
- **memformat**: not replaced
- **meminfo**: replaced by the storage command
- **outputformat**: replaced by the **instrument outputformat** command
- **pauseresume**: not replaced
- **permit**: not replaced.
- **power**: replaced by the **instrument power** command
- **powerexternal**: replaced by the **instrument power external** command
- **powerinternal**: replaced by the **instrument power internal** command
- **prompt**: replaced by the settings command
- **regimes**: replaced by the schedule command
- **regime**: replaced by the schedule command
- **sampling:** replaced by the schedule and deployment command
- **thresholding**: replaced by the **deployment gate** command
- **twisactivation**: replaced by the **deployment gate** command
- **readdata**: replaced by the **download** command
- **serial**: replaced by the **link serial** command
- **streamserial**: replaced by the schedule command
- **streamusb**: replaced by the schedule command
- **wifi**: not replaced

2.6.4 Improved commands and new parameters

- deployment
- enable
- disable
- verify
- channel
- calibration
- **instrument outputformat**
- settings
- sensor

2.6.5 New commands

For more information on the function and intention of these new commands see the individual command pages or see General overview:

- **config**
- dataset
- **factory**
- group
- instrument
- link
- parameters
- pcba
- poll
- schedule

2.7 Download stored data

Memory is organized onboard the instrument by datasets, then by schedules, then by data (samples), events and metadata.

The list of datasets available is obtained with the dataset command:

```
>> dataset
<< dataset count=3 maxcount=20 list=GullCove_aug22|GullCove_sep22|GullCove_oct22
```

The dataset command allows to list the different schedules available in a particular dataset:

```
>> dataset GullCove_aug22 schedulelist
<< dataset GullCove_aug22 schedulelist=sched_CTD|sched_ODO
```

Each schedule contains data, events, and metadata (refer to the dataset command). In order to download the data of one dataset's schedule, it is possible to check first the amount of data to be downloaded:

```
>> dataset GullCove_aug22/sched_CTD/data bytecount
<< dataset GullCove_aug22/sched_CTD/data bytecount=2608
```

Then download the data with the download command (here in chunks of 500 bytes):

```
>> download GullCove_aug22/sched_CTD/data bytecount=500 bytestart=0
<< download GullCove_aug22/sched_CTD/data bytecount=500 bytestart=0<cr><lf><bytes[0…
499]-of-data><CRC>

>> download
<< download GullCove_aug22/sched_CTD/data bytecount=500 bytestart=500<cr><lf><bytes[500…
999]-of-data><CRC>

>> download
<< download GullCove_aug22/sched_CTD/data bytecount=500
bytestart=1000<cr><lf><bytes[1000…1499]-of-data><CRC>

>> download
<< download GullCove_aug22/sched_CTD/data bytecount=500
bytestart=1500<cr><lf><bytes[1500…1999]-of-data><CRC>

>> download
<< download GullCove_aug22/sched_CTD/data bytecount=500
bytestart=2000<cr><lf><bytes[2000…2499]-of-data><CRC>

>> download
<< download GullCove_aug22/sched_CTD/data bytecount=108
bytestart=2500<cr><lf><bytes[2500…2607]-of-data><CRC>
```

⚠ The 2-bytes cyclic redundancy check should be ignored when parsing.

It is also possible to download the events recorded during the deployment:

```
>> dataset GullCove_aug22/events bytecount
<< dataset GullCove_aug22/events bytecount=480
>> download GullCove_aug22/events bytecount=500 bytestart=0
<< download GullCove_aug22/events bytecount=480 bytestart=0<cr><lf><bytes[0…479]-of-
events><CRC>
```

3 Commands

3.1 Communications

3.1.1 link

Usage

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

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

Security

Open.

Description

The link command provides information about the available communication links. When issued without the use of a sub command, the current communication link over which the command was received, is returned.

The communication link is returned as a **type** field. The possible responses are:

- usb
- serial

Examples

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

Here, communication is taking place over the **usb** link.

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

Here, communication is taking place over the **serial** link. To access parameters for a serial link, see the serial sub command.

3.1.1.1 serial

Usage

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

Security

Unsafe - no modifications while the instrument is enabled.

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.

Modifications to parameters may not be made while logging is enabled; this is to avoid disturbing any real-time data output.

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.

1. **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.
2. **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.
3. **uart**: This offers a 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).
4. **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. This value is only reported when explicitly requested.
- **availablebaudrates**: report the list of available baudrates. This value is only reported when explicitly requested.

Examples

```
>> link serial
<< link serial baudrate=19200 mode=rs232
```

Sending the serial command without any arguments will result in all parameters being returned. Note the absence of **availablemodes** and **availablebaudrates.**

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

Set the serial baudrate to 115200.

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

Request the serial mode, then set it to **rs485f**.

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

Request the **availablebaudrates**.

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

Request the **availablemodes**.

3.1.2 sleep

Usage

`>> sleep`

Security

Open.

Description

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

- Any interface circuitry used for a Serial link.
- Sensor channels activated *only* for the purpose of satisfying a poll 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 poll command will still be shut down.

If settings **confirmation=on** then the the sleep command will provide a confirmation response prior to performing the power saving measures, otherwise no response is expected.

Examples

```
>> settings confirmation
<< settings confirmation=off

>> sleep
% No response after issuing the sleep command
```

The sleep command is issued with the confirmation state = off. The instrument will not respond and will immediately move into a low power state.

```
>> settings confirmation
<< settings confirmation=on

>> sleep
<< sleep
```

The sleep command is issued with the confirmation state = on. The instrument will respond with the command prior to moving into a low power state.

3.2 Realtime data

3.2.1 poll

Usage

```
>> poll [ channellist=<channel_label_list...> ] |
[grouplist=<group_label_list...> ]
```

Security

Open.

Description

Requests an "on-demand" sample from the specified channel(s) in the instrument. If recent scheduled sample data is available for a channel, that value may also be returned to satisfy the **poll** request; "recent" in this context means less than one second old. If recent data is not available, a sample is explicitly acquired for the benefit of the **poll**. A sample acquired *only* for **poll** is never stored in memory. If the instrument is not actively logging, then all requested channels will be sampled explicitly for the **poll** request.

The instrument simply responds with the <*sample-data*>; depending on the configured settling time (or power-on settling delay) for the sensors sampled, there may be a noticeable delay before the <*sample-data*> appears. Refer to the **channel** command for details of access to the settling time of each channel.

The channel(s) to sample are specified by one of two methods; only one method may be used with any given instance of the **poll** command. Issuing the command without any channel specification has the same effect as specifying **poll channellist=<all_channel_labels>** - see below. The order in which the channel data is sent is the same as reported in response to the **channel** **channellist** command.

- **channellist**=<*channel_label_list…*>, specifies the channels to poll from. Labels in the list must be separated by a pipe character ("|"), with no spaces. If the desire is to specify all the channels then just use the **poll** command by itself without any argument. The order in which the channel data is sent is the same as reported by the **channellist.**

⚠ **channellist** can only contain either one channel label or the ‘all’ keyword on the L3.5 platform.

- **grouplist**=<*group_label_list…*>, specifies the groups of channels to be sampled. Labels in the list must be separated by a pipe character ("|"), with no spaces. Data will be returned for each group in the order they are specified. For a each group, the order in which the channel data is sent is the same as reported in response to the **group** <*group_label*> **chanellist** command. The specified groups must already exist; it can be one of the groups

created for scheduled sampling, or it can be any one of a number of groups created by the user specifically for polling operations.

grouplist can only contain one group on the L3.5 platform.

The output format of the <*sample-data*> is determined by the **instrument outputformat** command.

The schedule label ***polling*** will be displayed if **outputformat schedulelabel = on** has been configured.

Once the polling operation has completed the sensors are left powered on for eight (8) seconds; this is to avoid excessive power cycling when multiple **poll** commands are sent within a short time. If power is of concern it is recommended to use the sleep command following a poll to perform a controlled power down of the system.

If a channel appears multiple times in the poll request, the same data reading will be returned at all the appropriate locations for that channel.

```
>> poll channellist=pressure_00|conductivity_00|temperature_00|pressure_00
<< 2024-10-21 11:51:58.000 12.7049270 35.3154081 18.1742890 12.7049270
```

The reading for pressure_00 is returned at position 1 and position 4 as requested

```
>> group g_depth channellist
<< group g_depth channellist=pressure_00|seapressure_00|depth_00

>> group g_ctd channellist
<< group g_ctd channellist=conductivity_00|temperature_00|pressure_00

>> poll grouplist=g_depth|g_ctd
<< 2024-10-21 11:52:58.000 22.7035490 12.7035490 12.7035490 35.3154081 18.1742890
22.7049270
```

The reading for pressure_00 is returned at position 1 and position 6 as requested from the specified groups.

Examples

The exact format of the response is determined by properties set with the outputformat command. For clarity, most examples are shown with all options turned off, but there is one example to illustrate that the word **polling** is always used as the <*schedule_label*> if that property is enabled. This allows polled samples to be identified amongst a sequence of scheduled samples being streamed in real time.

These examples use the following conditions in the outputformat.

```
>> instrument outputformat
<< instrument outputformat sn=off schedulelabel=off datetime=on crc=off encoding=ascii
datatype=float32
```

```
>> poll
<< 2024-10-21 11:50:49.000 18.1745130 12.7052970 2.69308210
```

Poll a sample from all channels. Data will be returned in the order the channels are specified in ***channel list.***

```
>> poll channellist=temperature_00
<< 2024-10-21 11:50:55.000 18.1742890
```

Poll a single channel, `temperature_00`.

```
>> poll grouplist=g_depth
<< 2024-10-21 11:50:58.000 12.7049270 2.69273150
```

Poll a single group of channels. The output of the readings are based on the order of channels in ***group g_depth channellist.***

```
>> poll channellist=conductivity_00|temperature_00|pressure_00
<< 2024-10-21 11:51:58.000 35.3154081 18.1742890 12.7049270
```

Poll from a list of channels. Data will be returned in the order the channels are specified in the list.

```
>> poll grouplist=g_depth|g_ctd
<< 2024-10-21 11:52:58.000 12.7035490 2.69152730,35.3154081 18.1742890 12.7049270
```

Poll from a list of groups. In this example, data for *g_depth* will be returned, then data for *g_ctd*.

```
>> instrument outputformat
<< instrument outputformat sn=off schedulelabel=on datetime=on crc=off encoding=ascii
datatype=float32

>> poll grouplist=g_depth
<< polling 2024-10-21 11:53:58.000 12.7035490 2.69152730
```

Poll from a group when the <*schedule_label*> is enabled in the output formatting.

3.3 Instrument details

3.3.1 id

Usage

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

Security

Open.

Description

> This command is designed to be backward compatible with previous generations and, as such, does not conform to the Gen4 API.

This is a read-only command that reports basic information about the instrument:

- **model**, model name of the instrument; for example, **RBRconcerto4**.
- **version**, firmware version in <*major*>**.**<*minor*>**.**<*patch*>format; for example, **1.14.5**.
- **serial**, serial number is always reported using six digits, padded with leading zeroes if necessary; for example **092431**.
- **fwtype**, reports a firmware type code which depends on the product range that the instrument belongs to; for example, the code for an RBR*solo*^{4}, would be **130**.

Examples

```
>> id
<< id model = RBRoem, serial = 850032, version = 1.0.12, fwtype = 150
```

```
>> id serial
<< id serial = 850032
```

```
>> id model
<< id model = RBRoem
```

```
>> id version
<< id version = 1.0.12
```

```
>> id fwtype
<< id fwtype = 150
```

3.3.2 id4

Usage

```
>> id4 [model | sn | fwversion | fwtype ]
```

Security

Open.

Description

This is a read-only command that reports basic information about the instrument:

- **model**, model name of the instrument; for example, **RBRconcerto4**.
- **fwversion**, firmware version in <*major*>**.**<*minor*>**.**<*patch*>format; for example, **1.14.5**.
- **sn**, serial number is always reported using six digits, padded with leading zeroes if necessary; for example **092431**.
- **fwtype**, reports a firmware type code which depends on the product range that the instrument belongs to; for example, the code for an RBR*solo*[4], would be **130**.

Examples

```
>> id4
<< id4 model=RBRoem sn=850032 fwversion=1.0.12 fwtype=150
```

```
>> id4 sn
<< id4 sn=850032
```

```
>> id4 model
<< id4 model=RBRoem
```

```
>> id4 fwversion
<< id4 fwversion=1.0.12
```

```
>> id4 fwtype
<< id4 fwtype=150
```

3.3.3 instrument

Usage

```
>> instrument [ state | sn | model | pn | fwversion | fwtype | fwlock | datatype ]
```

```
>> instrument dump
```

```
>> instrument factory reset
```

```
>> instrument outputformat
```

```
>> instrument power [internal external]
```

```
>> instrument reboot
```

Security

Open.

Description

Reports some general information about the instrument:

- **state=**<*state*>: the state of the instrument. The possible states are:
  - **disabled**: The instrument has not been enabled. It is not actively running a deployment.
  - **enabled**: The enable command has been issued and the instrument is now running a deployment.
- **sn=**<*serial_number*> serial number is always reported using six digits, padded with leading zeroes if necessary; for example **092431**.
- **model**, model name of the instrument; for example, **RBRconcerto4**.
- **pn=**<*part_number*>: The RBR part number of the instrument
- **fwversion**, firmware version in <*major*>**.**<*minor*>**.**<*patch*>format; for example, **1.14.5**.
- **fwlock=**<*on*|*off*>: Indicates if firmware can be updated on the instrument or if it is locked to a specific version.
  - 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.
- **datatype** is the numeric format used to store data values in the memory for all channels. For normal deployments this will be either **float32** (IEEE single precision floating point) or **float64** (IEEE double precision floating point). This setting is factory configured; most instruments will use **float32**, but an instrument with very high precision sensors may use **float64** to maintain the necessary level of resolution.
  There is a third format used for calibration purposes, **calfloat64**; this format is the same numerically as **float64**, but no calibration equation is applied to the data. It is presented as a ratio compared to nominal full-scale, so the expected range is nominally 0.0 to 1.0. The full theoretical range is -2.0 to +2.0 but the output of most channels will remain within or close to the expected nominal range. The format **calfloat64** will be reported only *during* a deployment that was enabled using **storagemode=calibration**; refer to the **enable** command for more details.
- **fwtype=**<*firmware_type*> is a read-only parameter giving the firmware type code. Possible values are:
  - **120** for L3.5 instruments

Examples

```
>> instrument
<< instrument state=disabled sn=210000 model=RBRconcerto3 pn=L3-M13-F15-BEC12-INT12-
SCT16-SP11 fwversion=2.1.0 fwtype=120 fwlock=off datatype=float32
```

The instrument has not been enabled. It is not actively running a deployment. The firmware is not locked to a specific version.

```
>> enable config=profiling dataset=Trial_2
<< enable config=profiling dataset=Trial_2 storagemode=normal state=enabled
>> instrument
<< instrument state=enabled sn=210000 model=RBRconcerto3 pn=L3-M13-F15-BEC12-INT12-
SCT16-SP11 fwversion=2.1.0 fwtype=120 fwlock=off datatype=float32
```

The enable command has been issued and the instrument is now running a deployment. The firmware is not locked to a specific version.

```
>> instrument
<< instrument state=enabled sn=210000 model=RBRconcerto3 pn=L3-M13-F15-BEC12-INT12-
SCT16-SP11 fwversion=2.1.0 fwtype=120 fwlock=off datatype=float32
```

```
>> disable
<< disable state=disabled

>> instrument
<< instrument state=disabled sn=210000model=RBRconcerto3 pn=L3-M13-F15-BEC12-INT12-
SCT16-SP11 fwversion=2.1.0 fwtype=120 fwlock=off datatype=float32
```

The instrument was enabled, the *disable* command was issued and the instrument transitioned to the disabled state. The instrument is no longer running a deployment.

```
>> instrument
<< instrument state=disabled sn=210000 model=RBRconcerto3 pn=L3-M13-F15-BEC12-INT12-
SCT16-SP11 fwversion=2.1.0 fwtype=120 fwlock=on datatype=float32
```

The instrument has not been enabled. It is not actively running a deployment. The firmware is locked to a specific version so Ruskin will not be allowed to update it.

3.3.3.1 dump

Usage

`>> instrument dump`

Security

Open.

Description

Provides a multiline response with all configuration parameters stored in the instrument. The first response will contain a confirmation of the command with a count of the number of lines which will be returned. Each line will be terminated with a <CR><LF>. Only a single "Ready: " prompt (if enabled) appears after all output is complete. This prompt is not included in the lines count.

```
>> instrument dump
<< instrument dump lines=16
<< clock datetime=20000104075152 offsetfromutc=unknown
<< simulation state=off period=600000 channellist=conductivity_00|temperature_00|
pressure_00
<< deployment starttime=20230104000000 status=inactive gate=time simulation=off
<< storage used=1528 remaining=134216192 size=134217728
<< dataset count=3 maxcount=20 list=DeepCove|GullCove_sep22|GullCove_oct22
<< channel count=3 list=conductivity_00|temperature_00|pressure_00
<< channel conductivity_00 type=cond00 address=32 settlingtime=50 readtime=260
guardtime=20 userunits=mS/cm derived=off grouplist=none sensor=none
<< channel temperature_00 type=temp00 address=1 settlingtime=50 readtime=260
guardtime=20 userunits=C derived=off grouplist=none sensor=none
<< channel pressure_00 type=pres00address=2 settlingtime=50 readtime=260 guardtime=20
userunits=dbar derived=off grouplist=none sensor=none
<< calibration conductivity_00 equation=lin datetime=20171218175005
offset=0.0000000e+000 slope=1.0000000e+000 c0=9.9876543e+000 c1=7.5642301e+000
<< calibration temperature_00 equation=lin datetime=20171218175005 offset=0.0000000e+000
slope=1.0000000e+000 c0=9.9876543e+000 c1=7.5642301e+000
```

```
<< calibration pressure_00 equation=lin datetime=20171218175005 offset=0.0000000e+000
slope=1.0000000e+000 c0=9.9876543e+000 c1=7.5642301e+000
<< sensor count=3 list=rbr_conductivity_012345|rbr_pressure_987654|RBRtridente_567890
<< sensor rbr_conductivity_012345 sn=012345 pn=000000revA channellist=none
<< sensor rbr_pressure_987654 sn=987654 pn=000000revA channellist=none
<< sensor RBRtridente_567890 sn=567890 pn=000000revA channellist=none
<< Ready:
```

3.3.3.2 outputformat

Usage

`>> instrument outputformat [ sn | schedulelabel | datetime | crc | encoding | datatype ]`

Security

Open.

Description

Reports or sets properties of the format used to transmit data in real time over any communication link; this format applies to both polled data, and live streamed data if available.

If no arguments are given, all current property settings are reported.

The properties apply to all active schedules; different formats can not be simultaneously active in different situations.

The default format of the output is as follows:

- The output starts with a <*schedule_label*>; all information on this line applies to this schedule only.
- Next is a timestamp giving years, months, days, hours, minutes, seconds and thousandths (milliseconds), punctuated as shown in the example below.
- Then a value for each channel is 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 to ensure no loss of resolution with the **datatype** currently in force.
- All values are shown in 'engineering-notation', which is the same as scientific notation except that the exponents are constrained to be multiples of three.
- All elements are separated by a space.
- The line terminates with a <CR><LF> pair of characters.

Formally, the default format can be expressed as:

<*schedule_label*> YYYY-MM-DD hh:mm:ss.ttt <*value1*> <value2> ... <*valueN*><CR><LF>

Here is an example of the default format for a 3-channel logger:

sch_fast_CTD 2024-06-10 11:24:14.125 38.6671142e+000 22.0217124e+000 1.95962418e+003<CR><LF>

This default format may be modified by turning some properties on or off. The currently supported parameters are:

- **sn** [= **on** | **off**] determines whether or not the output begins with a preamble consisting of the string **RBR**, followed by a space, then the logger's 6-digit serial number. The default state is **off**.
- **schedulelabel** [= **on** | **off**] determines whether or not the <*schedule_label*> appears before the timestamp. The default state is **on**.

It it s recommended to set **schedulelabel**=**on** when multiple schedules are used for a deployment.

- **datetime** [= **on** | **off**] determines whether or not the timestamp appears. The default state is **on**.
- **crc** [= **on** | **off**] determines whether or not a Cyclic Redundancy Check (CRC) is included at the end of the line immediately before the terminating <CR><LF> pair. The default state is **off**.
  The CRC includes all characters already sent on this line, starting with the first, up to and including the last space character before the <*CRC*>. The calculation uses the 16-bit CCITT polynomial, f(x)=x^16 + x^12 + x^5 + 1, feeding each byte into the generator least significant bit first, and using 0xFFFF as the seed value. The format of the reported CRC is **0xHHHH**, where HHHH are four hexadecimal digits; the **0x** part of the string is *not* included in the CRC.
- **encoding** [**= ascii** | **binary**] determines whether the output is sent in a human-readable **ascii** form such as that shown in the example above, or a more compact **binary** format that is more machine-readable for easier parsing.

The **binary** option is provided only for compatibility with the command set of other instruments. The **encoding** is always set to **ascii**, and attempting to change it will provoke an error message. The default setting is **ascii**.

- **datatype** [**= float32** | **float64**] | [= **calfloat64**] is the numeric format used to report data values for all channels. For normal deployments this will be either **float32** (IEEE single precision floating point) or **float64** (IEEE double precision floating point), and the end user may specify either option. However, when the instrument's native data type is **float32**, specifying **datatype**=**float64** will *not* improve the actual precision of the values; there will just be extra meaningless digits in the output. On the other hand, if the native data type is **float64**, specifying **datatype**=**float32** will result in fewer bytes being transmitted, which may be useful for sending reduced resolution data over a slow (or expensive) telemetry channel. The instrument's native data type is set at the factory so that it is appropriate for the installed sensors, and can not be changed; see **storage datatype**. The setting of **outputformat datatype** should match the native data type when shipped from the factory. Changing **outputformat datatype** does not affect the resolution of data stored in memory.
  For a deployment enabled in calibration mode (see **enable storagemode=calibration**), the **outputformat datatype** reported will be **calfloat64**, regardless of what the instrument's native data type is. This uses IEEE double precision floating point, but reports unprocessed values as a proportion of full scale in the nominal range -1.0 to 1.0. This setting can not be made directly using the **outputformat** command; it is a result of the options used with the **enable** command. This setting will persist only as long as such a deployment is active; when the deployment finishes, the setting reported will revert to either **float32** or **float64**, whichever was in force prior to the calibration deployment.

All properties may be set independently from one another; they may be used singly or in combinations. Refer to the examples below for usage and the impact on the streamed data output.

Examples

```
>> instrument outputformat
<< instrument outputformat sn=off schedulelabel=on datetime=on crc=off encoding=ascii
datatype=float32
```

These are the default settings that produce the default format.

```
>> instrument outputformat schedulelabel=off
<< instrument outputformat schedulelabel=off
```

Remove from the default format the schedule label, to simplify the output when only one schedule is used:

Format:
YYYY-MM-DD hh:mm:ss.ttt <*value1*> <*value2*> ... <*valueN*><CR><LF>

Three-channel logger example:
2024-06-10 11:24:14.125 38.6671142e+000 22.0217124e+000 1.95962418e+003<CR><LF>

```
>> instrument outputformat sn=on
<< instrument outputformat sn=on
```

Add to the default format the preamble that includes the serial number:

Format:
RBR <*serial_number*> <*schedule_label*> YYYY-MM-DD hh:mm:ss.ttt <*value1*> <*value2*> ... <*valueN*><CR><LF>

Three-channel logger example:
RBR 142152 sch_fast_CTD 2024-06-10 11:24:14.125 38.6671142e+000 22.0217124e+000 1.95962418e+003<CR><LF>

```
>> instrument outputformat crc=on
<< instrument outputformat crc=on
```

Add to the default format the CRC, for applications where high confidence in output correctness is required:

Format:
<*schedule_label*> YYYY-MM-DD hh:mm:ss.ttt <*value1*> <*value2*> ... <*valueN*> <*CRC*><CR><LF>

Three-channel logger example:
sch_fast_CTD 2024-06-10 11:24:14.125 38.6671142e+000 22.0217124e+000 1.95962418e+003 0xB9D8<CR><LF>

3.3.3.3 power

Usage

```
>> instrument power [ source ]
>> instrument power internal [ voltage | batterytype | capacity | used ]
>> instrument power external [ voltage | batterytype | capacity | used ]
```

Security

Unsafe.

Description

Reports parameters or executes sub-commands relating to the logger's power sources as follows:

- **source** is a read-only parameter which responds with one of the following names:
  - **usb** the logger is drawing power from the USB connection.
  - **internal** the logger is drawing power from its internal battery.
  - **external** the logger is drawing power from an external power source.
- **internal** is a sub-command used to access all the available parameters associated with the logger's internal battery.
- **external** is a sub-command used to access all parameters associated with the logger's external power supply.

The source parameter can not be used together with sub-commands in a single instance of the command, and only one sub-command at a time can be invoked. A sub-command must immediately follow the **instrument power** command. Parameters of a sub-command must follow the sub-command. Here are some examples of valid and invalid commands; invalid commands will provoke an error message:

- ✅ **instrument power**
- ✅ **instrument power source**
- ✅ **instrument power external**
- ✅ **instrument power internal capacity**
- ❌ **instrument power source external**
- ❌ **instrument power external source**
- ❌ **instrument power internal external**
- ❌ **instrument power voltage internal**

For more details on the sub-commands, refer to the specific internal and external option pages.

Examples

```
>> instrument power source
<< instrument power source=internal
```

```
>> instrument power internal
<< instrument power internal voltage=12.39 batterytype=nimh capacity=138.000e+003
used=93.170e+003
```

3.3.3.3.1 external

Usage

```
>> instrument power external [voltage | batterytype | capacity | used ]
```

Security

Unsafe - parameters can not be changed if the instrument state is enabled.

Description

Allows various parameters associated with the external power supply to be reported or set.

- **voltage** is a read-only parameter giving a live measurement of the voltage detected by the logger at its external supply input connection.
- **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** can not 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)*
    - **other**
    - **none**
- **capacity** is a read-only parameter which reports the total nominal energy capacity (in Joules) of the external battery pack. It can not be changed directly, but changes according to the selected **batterytype**. The capacity value for the power source types **other** and **none** is currently zero, but energy used from these sources can still be tracked.
- **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

```
>> instrument power external
<< instrument power external voltage=14.21 batterytype=fermata_lisocl2
capacity=22.000e+006 used=100.100e+003
```

```
>> instrument power external used=0
<< instrument power external used=0.000e+000
```

```
>> instrument power external batterytype=fermata_lisocl2
<< instrument power external batterytype=fermata_lisocl2
```

3.3.3.3.2 internal

Usage

```
>> instrument power internal [ voltage | batterytype | capacity | used ]
```

Security

Unsafe - parameters can not be changed if the instrument state is enabled.

Description

Allows various parameters associated with the internal battery to be reported or set.

- **voltage** is a read-only parameter giving a live measurement of the voltage detected by the logger at the internal battery terminals.
- **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** can not 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)
    - **other**
    - **none**
- **capacity** is a read-only parameter which reports the total nominal energy capacity (in Joules) of the internal battery selected. It can not be changed directly, but changes according to the **batterytype**. The capacity value for the power source types **other** and **none** is currently zero, but energy used from these sources can still be tracked.
- **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

```
>> instrument power internal
<< instrument power internal voltage=6.52 batterytype=nimh capacity=138.000e+003
used=100.100e+003
```

```
>> instrument power internal used=0
<< instrument power internal used=0.000e+000
```

```
>> instrument power internal batterytype=lifes2
<< instrument power internal batterytype=lifes2
```

3.3.3.4 reboot

Usage

```
>> instrument reboot [delay=<milliseconds-delay>]
```

Security

```
Open
```

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.

Whether or not there is a response to the command depends on the **confirmation** setting: if confirmation is off there will be no response - see the examples below. The **confirmation** setting is controlled by the settings command.

The **delay** option can be useful when using the command over a USB-CDC communication link. A *<milliseconds-delay>* value must be supplied if the option is used; there is no default value. 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. Not all communications software is able 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.

Examples

```
% settings confirmation=on
>> instrument reboot
<< instrument reboot
% reboot ocurrs
```

Successfully resets the logger CPU following the confirmation of the request.

```
% settings confirmation=on
>> instrument reboot delay=5000
% ... 5-second delay...
<< instrument reboot delay=5000
% reboot ocurrs
```

Successfully resets the logger CPU following the requested delay. Confirmation of the request is sent if confirmation is enabled, but of course if the purpose of the delay was to allow time to disconnect the logger, the message will not be seen.

```
% settings confirmation=off
>> instrument reboot
% reboot occurs
```

Successfully resets the logger CPU without issuing a confirmation of the request.

3.3.4 pcba

Usage

```
>> pcba [ count | list ]
```

```
>> pcba <pcba_label> [ sn | pn | fw | hw | address ]
```

Security

Open.

Description

This command is used to access information regarding all the pcba instances configured within the instrument. **Pcba**s represent physical circuit cards and provide information which helps when debuging is necessary. **Pcba** information is read-only.

To access general information about all pcbas within an instrument the command can be sent without any arguments. The parameters which will be returned are:

- **count** is the number of pcbas installed in the instrument.
- **list** reports a list of all the pcba labels. There is no particular significance to the order in which items are reported, but for a given instrument the order is fixed. Labels in the list are separated by a pipe character (“**|**”), with no spaces.

```
>> pcba
<< pcba count=3 list=1352460_00|0123456_00|6543210_00
```

To access information for a specific **pcba**, send the `<pcba_label>` as an argument to the `pcba` command. The following parameters provide the basic information available for all **pcba**s. They are all read-only parameters.

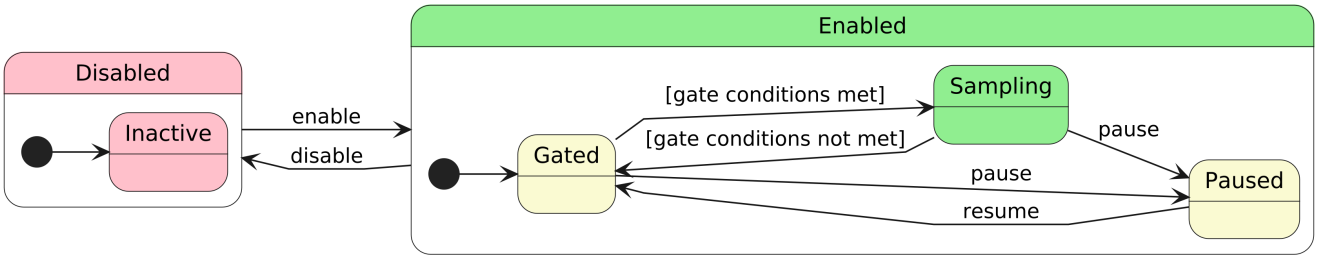
- **sn** reports the serial number of the pcba
- **pn** reports the **pcba**'s part number
- **fw** reports the **pcba**'s current firmware version
- **hw** reports the hardware revision
- **address** reports the base address used to communicate with the **pcba** over the bus

```
>> pcba 0123456_00
<< pcba 0123456_00 sn=123456 pn=0123456revA fw=1.1.1 hw=A01 address=128
```

3.4 Deployments

An instrument can be **enabled** for one deployment at a time; as such, the instrument state is either **enabled** or **disabled**, and the deployment state is one of **gated**, **sampling**, or **paused** if the instrument is **enabled**, or the deployment state is **inactive** if the instrument is **disabled**. See also enable, disable, pause and resume.

Following is the state diagram for the instrument/deployment pair:



3.4.1 clock

Usage

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

Security

Unsafe.

Description

Retrieve or set the logger's current date and time. The clock can only be changed when the instrument is not logging or streaming.

- **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. Any change made is persistent, and will be retained until changed again. By default the `offsetfromutc` **+0.00** which represent UTC.

Examples

```
>> clock
<< clock datetime=20240401120000 offsetfromutc=+1.00
```

```
>> clock datetime=20240401120130
<< clock datetime=20240401120130

>> clock
<< clock datetime=20240401120130 offsetfromutc=+1.00
```

```
>> clock offsetfromutc=-4.00
<< clock offsetfromutc=-4.00

>> clock
<< clock datetime=20240401120130 offsetfromutc=-4.00
```

3.4.2 verify

Usage

```
>> verify config=<configuration_label> dataset=<dataset_label>
[ storagemode=normal | calibration ]
```

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 instrument state, a warning or an error message. It does not, however, actually enable the instrument for sampling.

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

The required parameters are as follows:

- **config=**<*configuration_label*> specifies the configuration which will define this deployment. The configuration must be valid.
- **dataset=**<*dataset_label*> gives the deployment's dataset a user-specified label. The <*dataset_label*> must satisfy all **naming constraints**.

The labels for all datasets existing in the logger's memory at any given time must be unique. A <*dataset_label*> may be reused, but only if the associated dataset is deleted from memory first. The <*dataset_label*> can not be changed after the **enable** command has been executed; if it is important, choose carefully.

There is also one optional parameter:

- **storagemode=normal** | **calibration** determines whether calibration equations will be applied to all channel data (**normal**), or not (**calibration**). The setting applies *only* to the current deployment, and will default to **normal** if not specified. When **storagemode=calibration**, all data values are stored as IEEE double precision floating point numbers in the nominal range 0.0 to 1.0, regardless of what the **normal** storage format is.

If any of the deployment consistency checks fail, an error message is sent. The most severe error found causes immediate failure of the command; a single attempt to verify the configuration will not detect multiple errors.

If successful, the command reports these parameters in its response:

- **config=**<*configuration_label*>, confirming the configuration that will be used for this deployment.
- **state=**<*instrument_state*> the instrument state which *would be assumed* by the instrument if the **enable** command were issued. The actual state of the instrument does not change.
- **dataset=**<*dataset_label*>, confirming the label of the dataset for this deployment.
- **storagemode=normal** | **calibration**, confirming the data storage mode to be used.

The command may succeed, but have a warning to report; in such a case the warning code appears at the start of the response. Refer to the examples below.

Examples

```
>> verify config=profiling dataset=test
<< verify config=profiling dataset=test storagemode=normal state=enabled
```

The programmed schedules are valid and the instrument would be enabled. The label **test** is a valid dataset label.

```
>> verify dataset=Trial_2 config=tides
<< verify config=tides dataset=DeepCove storagemode=normal state=enabled
```

The programmed schedules are valid and the instrument would be enabled. The label **Trial_2** is a valid dataset label.

```
>> instrumemnt state
<< instrument state=enabled
>> verify dataset=DeepCove config=tides
<< WRN-408 instrument was already enabled
```

The instrument is already in an enabled state and using the same configuration as what is specified.

```
>> instrumemnt state
<< instrument state=enabled
>> verify dataset=ShallowCove config=tides
<< ERR-128 instrument was already enabled with different settings
```

The instrument is already in an enabled state using settings which do not match the input. The existing deployment will continue on as it was. Attempting to enable using these parameters will provoke the same error.

3.4.3 enable

Usage

```
>> enable config=<configuration_label> dataset=<dataset_label>
[ storagemode=normal | calibration ]
```

Security

Open.

Description

Enables the logger to sample for a new deployment according to the specified configuration. The following parameters are required:

- **config=**<*configuration_label*> specifies the configuration which will define this deployment. The configuration must be valid.
- **dataset=**<*dataset_label*> gives the deployment's dataset a user-specified label. The <*dataset_label*> must satisfy all **naming constraints**.

The labels for all datasets existing in the logger's memory at any given time must be unique. A <*dataset_label*> may be reused, but only if the associated dataset is deleted from memory first. The <*dataset_label*> can not be changed after the **enable** command has been executed; if it is important, choose carefully.

There is also one optional parameter:

- **storagemode=normal** | **calibration** determines whether calibration equations will be applied to all channel data (**normal**), or not (**calibration**). The setting applies *only* to the current deployment, and will default to **normal** if not specified. When **storagemode=calibration**, all data values are stored as IEEE double precision floating point numbers in the nominal range 0.0 to 1.0, regardless of what the **normal** storage format is.

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. The most severe error found causes immediate failure of the command; a single attempt to enable the logger will not detect multiple errors. To determine in advance whether the command should succeed, use the verify command to perform a dry run first.

If all required conditions are satisfied, the **enable** command may still fail in the event of a fault that prevents logging from being enabled; this also will provoke an error message.

If successful, the command reports these parameters in its response:

- **config=**<*configuration_label*>, confirming the configuration that will be used for this deployment.
- **state=**<*instrument_state*> the state of the instrument; this <*instrument_state*> is also reported by the instrument command.
- **dataset=**<*dataset_label*>, confirming the label of the dataset for this deployment.
- **storagemode=normal** | **calibration**, confirming the data storage mode to be used.

The command may succeed, but have a warning to report; in such a case, the warning code appears at the start of the response. Refer to the examples below.

Examples

```
>> enable config=profiling dataset=test
<< enable config=profiling dataset=test storagemode=normal state=enabled
```

The programmed schedules are valid and the instrument has been enabled. The dataset has been assigned the user-supplied label **test**.

```
>> enable dataset=Trial_2 config=tides
<< enable config=tides dataset=Trial_2 storagemode=normal state=enabled
```

The programmed schedules are valid and the instrument has been enabled. The dataset has been assigned the user-supplied label **Trial_2**.

```
>> enable config=pH_cal dataset=d_pHcal_20240401 storagemode=calibration
<< enable config=pH_cal dataset=d_pHcal_20240401 storagemode=calibration state=enabled
```

The programmed schedules are valid and the instrument has been enabled; the calibration **storagemode** is being used for calibration of a sensor. The dataset has been assigned the user-supplied label **d_pHcal_20240401.**

```
>> instrument state
<< instrument state=enabled
>> enable config=pH_cal dataset=d_pHcal_20240401 storagemode=calibration
<< WRN-408 instrument was already enabled
```

The instrument was already enabled with those exact settings. The deployment will continue on as it was.

```
>> instrument state
<< instrument state=enabled
>> enable config=pH_cal dataset=d_pHcal_20240401 storagemode=calibration
<< ERR-128 instrument was already enabled with different settings
```

The instrument was already enabled with settings which don't match the input. The existing deployment will continue on as it was. The new deployment will not be enabled.

3.4.4 disable

Usage

```
>> disable
```

Security

Open.

Description

If the instrument is logging, this command will terminate the current deployment. If the instrument is not logging when the command is issued, it will respond with a warning message but takes no other action; its state does not change. When successful, the command reports:

- **state**, the state of the instrument is always reported when the command is complete (see also the **Instrument** 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 instrument's

response to the command may be delayed. If the instrument is sampling in any averaging or burst recording mode, the burst currently in progress will be interrupted and abandoned.

If the instrument is recording data to memory, a “stop event” will be appended to the data after the last sample stored.
Examples

```
>> instrument
<< instrument state=enabled

>> disable
<< disable state=disabled
```

The instrument was enabled, and has now been disabled.

```
>> instrument
<< instrument state=disabled

>> disable
<< WRN-435 instrument state is already disabled
```

The instrument had previously been disabled; its status has not changed.

3.4.5 deployment

Usage

```
>> deployment [ starttime | status | gate | simulation ]
```

Security

Unsafe - modifications are not permitted while logging is enabled.

Description

Allows various parameters to be reported or set.

- **starttime** [=<*YYYYMMDDhhmmss*>], retrieve or set the start date and time of the next deployment. Available only when *gate=time*
- **status** is a read-only parameter which returns the current state of the finite state machine for the instrument's sampling function. Possible values are given below:
  - **sampling** : deployment is active.
  - **gated** : deployment is active but waiting for a gating condition to be met to start or resume sampling.
  - **paused** : deployment is active but the pause command has been issued so sampling has been paused.
  - **inactive** : deployment is inactive. The instrument state is disabled.
- **gate**=<gate_condition> retrieve or set a gating condition of the next deployment currently enabled. If a deployment is currently active then the value is read-only.
  A **gate** is an extra requirement that must be satisfied before sampling will occur. The following gating conditions are presently defined:
  - **none**: no gating conditions are enabled. The instrument will start sampling once enabled.
  - **time**: the instrument will start sampling once its clock is after deployment starttime.
- **simulation**=**on** | **off** is a read only parameter indicating whether *any* of the instrument's channels are being simulated (**on**) or whether they are all reporting true measured data (**off**). Refer also to the simulation command for control of other parameters.

Examples

```
>> deployment
<< deployment starttime=20230104000000 status=inactive gate=time simulation=off
```

```
>> deployment starttime=20230104000000
<< deployment starttime=20230104000000
```

```
>> deployment status
<< deployment status=inactive
```

```
>> enable config=default_config dataset=dataset_01
<< enable config=default_config dataset=dataset_01 storagemode=normal state=enabled

>> deployment
<< deployment status=sampling gate=none simulation=off
```

3.4.6 pause

- **Usage**

`>> pause`

Security

Open.

Availability

v2.2.0

Description

This command pauses an enabled deployment.

If successful, the command reports its status:

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

The only error condition which would prevent the successful execution of the command is if the instrument had not been enabled for deployment.

When a pause is issued, the following will happen:

- Streaming is immediately suspended if any schedule in the deployment was configured to do so.
- 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. In regimes mode, any open bin will be closed.
- An event of type EVENT_PAUSE (0x2B) is stored in the active dataset.
- 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 acquisition, if any, finishes.

```
>> pause
<< ERR-406 cannot 'pause' while not logging
```

Cannot pause deployment if the instrument has not been enabled.

3.4.7 resume

Usage

```
>> resume
```

Security

Open.

Availability

v2.2.0

Description

This command 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:
  - **sampling**
  - **gated**

The only error conditions which would prevent the successful execution of the command are if the instrument has not be enabled for deployment, or was not previously paused.

When a resume is issued, the following will happen:

1. Streaming is immediately activated if any schedule in the deployment was configured to do so.
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 active dataset.
4. Acquisitions will continue to be scheduled at normal intervals.

Examples

```
>> resume
<< resume status=gated
```

Deployment has resumed running as scheduled and is currently waiting on a gating condition.

```
>> resume
<< resume status=sampling
```

Deployment has resumed running as scheduled.

```
>> resume
<< ERR-407 cannot 'resume' unless paused
```

Cannot resume a deployment unless the instrument has already been enabled and then paused.

3.4.8 simulation

Usage

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

Security

Unsafe.

Description

This command controls how the simulation mode will operate, if used. It is an Unsafe command, and as such can not be modified while the instrument is enabled. It takes as parameters:

- **state=on** | **off** determines whether the channels specified in the **channellist** option will be simulated (**on**), or report true measured data (**off**). The default setting as shipped from the Factory is **off**. When changed, the setting is persistent; it will remain in force from one deployment to the next.
- **period** [=<*milliseconds*>] is the period in milliseconds of one full cycle of simulated values. The default value is 3600000ms, corresponding to 1 hour.
- **channellist** [=<*list_of_channel_labels*>] specifies which channels will be simulated. Each channel is specified by its label, and channel labels in the list are separated by a pipe character ('**|**') with no spaces. The order in which channels are specified does not matter.
  Derived channels can not 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 <*list_of_channel_labels*>; **all** selects all measured channels for simulation, while **none** can be used to disable simulation for all channels.

During a deployment, the active simulation state can be requested using the command **deployment simulation**.

An obvious danger of using the persistent setting **state=on** is that it would be possible to inadvertently enable an instrument for a real deployment with one or more channels simulated instead of measuring real data, so use this option with care.

When simulation is enabled, 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 [1]	dbar
conductivity	-1	+85	mS / cm
PAR	-25	+2500	µmol / m^2 / s
turbidity	-25	+2500	NTU
chlorophyll	-2	+150	µg / L
O_2 concentration	0	+450	µM
All other types	25% full scale	75% full scale	as appropriate

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

The simulated values are used both for scheduled samples (whether stored in memory, streamed, or both) and for on-demand samples obtained using the **poll** command. Both types of sample should conform 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.

Examples

```
>> simulation
<< simulation state=off period=600000 channellist=conductivity_00|temperature_00|
pressure_00

>> simulation period=3600000
<< simulation period=3600000

>> simulation channellist=conductivity_00|temperature_00|pressure_00|chlorophyll_00
<< simulation channellist=conductivity_00|temperature_00|pressure_00|chlorophyll_00

>> simulation state=on
<< simulation state=on
```

3.5 Memory and datasets

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 dataset

Usage

```
>> dataset [ count | maxcount | list ]
```

```
>> dataset <dataset_label> [ status | schedulelist | bytecount | datatype ]
```

```
>> dataset <dataset_label>/[<schedule_label>/]<block>
```

```
>> dataset delete <dataset_label> | all
```

Security

Open.

Description

This command is used to access information regarding all of the datasets stored in the instrument. Datasets contain a list of schedules to be run in a deployment. The configuration is specified at the time of enabling the instrument. A dataset is created automatically when the instrument is enabled. Datasets cannot be modified but they can be deleted.

To access general information about all of the datasets stored within the instrument use the `dataset` command without any arguments. The following list of parameters will be returned:

- **count** reports the number of datasets currently stored in the instrument's memory.
- **maxcount** reports the maximum number of datasets that the instrument can store in its memory.
- **list** reports a list of the labels assigned to the datasets; the labels are separated by a pipe character ('|'), with no spaces. Labels are reported in the order that the datasets were created, earliest first.

```
>> dataset
<< dataset count=3 maxcount=20 list=DeepCove|GullCove_sep22|GullCove_oct22
```

If there are no datasets stored in the instrument the following response will be observed.

```
>> dataset
<< dataset count=0 maxcount=20 list=none
```

To access parameters for a specific dataset, provide the `<dataset_label>` as an argument to the `dataset` command. The following parameters are available for a given dataset:

- **status** reports one of two values:
  - **open**, if the dataset is for a deployment currently in progress, or
  - **closed**, for a historical dataset in memory which is no longer being updated because its deployment has stopped.
- **schedulelist** reports a list of the schedules executed during the deployment associated with the specified dataset; these labels will identify the retrievable sample storage objects. It is important to realize that after a deployment has finished, schedules may be edited, renamed or deleted; this option reports the schedules as they were when the specified dataset was started, not as they are presently defined in the logger.
- **bytecount** reports the total amount of memory used by this dataset, in bytes. For more detailed information about how this total is broken down, use the second form of the command.
- **datatype** reports the numerical format used for data storage in this dataset; one of **float32**, **float64** or **calfloat64**; see also the storage and enable commands.

```
>> dataset DeepCove
<< dataset DeepCove status=closed schedulelist=tides_schedule|DO_schedule
bytecount=3749498 datatype=float32
```

The second form of the command reports read-only information about the specified dataset's usage of memory. It can be used to find out how much memory is used for sample data, events or metadata, either for the whole dataset or on a per-schedule basis. The general form of the command is:

```
>> dataset <dataset_label>[/<schedule_label>][/<block>]
<< dataset <dataset_label>[/<schedule_label>][/<block>] bytecount=<block_bytes>
<count_key>=<count_value>...
```

- <*dataset_label*> is always required, and identifies the dataset of interest, For a list of available datasets, use **dataset list**.
- <*schedule_label*> is optional, although at least one of <*schedule_label*> or <*block*> must be provided. If <*schedule_label*> is given, the reported information applies to the named schedule only. If <*schedule_label*> is omitted, the reported information is the sum for all schedules in the dataset. For a list of available schedules, use **datasets** <*dataset_label*> **schedulelist**.
- <*block*> is optional, although at least one of <*schedule_label*> or <*block*> must be provided. If <*block*> is given, it must be one of the three keywords **data**, **events**, or **meta**; it determines the type of information retrieved for the specified schedule. If <*block*> is omitted, all three types of information will be reported.
    - **data**, to report memory usage for sample data.
    - **events**, to report memory usage for events.
    - **meta**, to report this schedule's memory usage for metadata.

When reporting about **events** or **meta**, a <*schedule_label*> should not be specified; these types of data can not be retrieved on a per-schedule basis, only for the entire deployment. Use one of <*dataset_label*>**/events** or <*dataset_label*>**/meta**.

It is important to understand that the '**/**' character is not just a separator; although it marks the boundaries between components, it also combines them into a single parameter. ensuring that they appear in the correct order with no other parameters in between.

The elements in the response to the command are as follows:

- **bytecount =** <*block_bytes*> is always reported, and gives the size in bytes of the requested block (**data** / **events** / **meta**).
- <*count_key*> **=** <*count_value*> is specific to the requested block, and gives the size in more 'natural' units than bytes:
    - for **data**, the form is **samplecount =** <*number_of_samples*>.
    - for **events**, the form is **eventcount =** <*number_of_events*>.
    - For metadata, there is no underlying 'natural' measure of size, so only the **bytecount** is reported.

```
>> dataset DeepCove/DO_schedule
<< dataset DeepCove/DO_schedule bytecount=3501264 samplecount=291106 eventcount=498
```

```
>> dataset DeepCove/DO_schedule/data
<< dataset DeepCove/DO_schedule/data bytecount=3493272 samplecount=291106
```

3.5.1.1 delete

Usage

```
>> dataset delete <dataset_label> | all
```

Security

Open.

Description

Datasets can be deleted using the **delete** action within the **dataset** command. A `<dataset_label>` must be specified as an argument to the action. Only a single dataset can be deleted at a time. The only exception to that is when the `all` argument is provided as the `<dataset_label>`. This will cause all datasets to be deleted.

```
>> dataset
<< dataset count=3 maxcount=20 list=DeepCove|GullCove_sep22|GullCove_oct22

>> dataset delete DeepCove
<< dataset delete DeepCove

>> dataset
<< dataset count=2 maxcount=20 list=GullCove_sep22|GullCove_oct22

>> dataset delete all
<< dataset delete all

>> dataset
<< dataset count=0 maxcount=20 list=none
```

3.5.2 download

Usage

```
>> download <dataset_label>/<schedule_label>/<block> <count_key>=<count_value>
<start_key>=<start_offset>
```

Security

Open.

Description

Reads all or part of a specified storage object. For the initial use of the **download** command, all parameters are required. When retrieving large amounts of information in 'blocks' using multiple instances of the command, parameters can be optional; this is discussed following the description of the parameters below.

- *<dataset_label>***/***<schedule_label>***/***<block>* specifies the storage object to be retrieved. The specifier has three components:
    - *<dataset_label>* identifies the complete dataset corresponding to the deployment of interest, For a list of available datasets, use **dataset list**.
    - *<schedule_label>* identifies the sampling schedule used to acquire the data. For a list of available schedules, use **dataset** *<dataset_label>* **schedulelist**.

- <*block*> is one of the three keywords **data**, **events**, or **meta**, and determines the type of data retrieved for the specified schedule.

It is important to understand that the "**/**" character is not just a separator; although it marks the boundaries between specification components, it also combines them into a single parameter. ensuring that they appear in the correct order with no other parameters in between.

When a source is specified, the <*dataset_label*> and <*block*> are required components; the instrument must know which dataset to read and what type of information is required. When retrieving a <*block*> of sample **data**, a <*schedule_label*> is also required; it is not possible to download sample data from multiple schedules with a single instance of the command.

When retrieving **events** or **meta**, a <*schedule_label*> should not be specified; these types of data can not be retrieved on a per-schedule basis, only for the entire deployment. Use one of <*dataset_label*>**/events** or <*dataset_label*>**/meta**.

- <*count_key*> = <*count_value*> specifies the quantity of information that the instrument should attempt to retrieve and report. As well as introducing the <*count_value*>, the <*count_key*> also determines the 'units' in which the information will be measured:
  - The **bytecount** <*count_key*> indicates that the information is measured in bytes; this option is available for all three block types, **data**, **events,** and **meta**.
  - The **samplecount** <*count_key*> can be used only with the **data** block type, and indicates that the information is measured in samples.
  - The **eventcount** <*count_key*> can be used only with the **events** block type, and indicates that the information is measured in events.

  Any transfer from a given storage object that uses multiple **download** commands must all use the same measurement units.

- <*start_key*>**=**<*start_offset*> specifies an offset from the beginning of the storage object where reading should begin. It must be specified in the same units used for the <*count_key*>; one of bytes, samples, or events:
  - **bytestart** indicates that the starting point is specified in bytes.
  - **samplestart** indicates that the starting point is specified in samples.
  - **eventstart** indicates that the starting point is specified in events.

  The first item stored in memory for this deployment always has <*start_key*>**=0**.

In all cases, if the requested amount of data would overrun the boundary of the target object, a valid transfer still occurs, but the amount of data actually returned will be less than the request. This is reflected in the instrument's response to the command.

The instrument responds with all parameter values it intends to use with this instance of the command. As with most commands, this response is terminated by a <*carriage_return*><*line_feed*> pair of characters. For example:

>> **download** <*dataset_label*>**/**<*schedule_label*>**/data samplecount**=<*requested_sample_count*>**,**
samplestart=<*start_sample_offset*>
<< **download** <*dataset_label*>**/**<*schedule_label*>**/data bytecount =** <*expected_byte_count*>**,**
samplecount=<*expected_sample_count*>**, samplestart=**<*start_sample_offset*>

The requested data then follows immediately, in binary format as it is stored in the instrument. Any multi-byte quantities are stored and transmitted least-significant-byte first.

Note that the count measurements reported in the response depend in part on how the retrieval was specified.

- **bytecount** is a valid <*count_key*> for all three <*block*> types specified in the **source**, namely **data**, **events,** and **meta**. It is always reported in the response, and will always accurately reflect the number of bytes the instrument is attempting to return. This assists the host in knowing exactly how much data to expect, even if the count was specified in samples or events. Note that the 2-byte CRC appended to the transfer is NOT included in

this count; the count indicates the quantity of information being returned from the logger's memory, and does not take into account the transfer protocol used.

- **samplecount** is a valid <*count_key*> only for <*block*> type **data**. It will be reported in the response only if specified in the command. For <*block*> types **events** and **meta** a **samplecount** is not valid and is never reported.
- **eventcount** is a valid <*count_key*> only for <*block*> type **events**. It will be reported in the response only if specified in the command. For <*block*> types **data** and **meta** an **eventcount** is not valid and is never reported.

If **data** or **events** are requested using **bytecount**, the instrument can not guarantee that this will correspond to a whole number of objects (samples or events). If that is a requirement, use **samplecount** or **eventcount** instead; the **bytecount** reported in the response will then be accurate for the requested number of objects.

At the end of the data the instrument appends a CRC (cyclic redundancy check). This 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. The bytes of the CRC computed by the instrument are swapped before appending to the data; this means that the host can include them in its CRC-check as an extra two bytes, and if the CRC is correct this always gives a result of zero. These two bytes are not included in the **bytecount** reported in the command's response.

Using multiple instances of the command

When retrieving a large amount of information from the instrument, for the purpose of managing potential communications errors it is advisable to break the transfer into smaller amounts using multiple instances of the **download** command.

Transfers specified in bytes always use the byte values for these substitutions and calculations, regardless of whether they represent whole numbers of objects. For transfers specified in samples or events the correspondence between whole numbers of objects and bytes is exact.

Examples

```
>> download dataset_00/sched_CTD/data bytecount=32000 bytestart=0
<< download dataset_00/sched_CTD/data bytecount=32000 bytestart=0
<< <32000 bytes of sample data from the start of the specified schedule><CRC>

    % The 1st part of a transfer of sample data from the schedule labelled sched_CTD,
starting with the first byte.
```

```
>> download dataset_00/meta bytecount=64000 bytestart=0
<< download dataset_00/meta bytecount=13584 bytestart=0
<< <13584 bytes of metadata associated with the entire dataset><CRC>
    % A new request for all the metadata associated with the dataset.  Again the number
of bytes available is less than the requested number.  The <schedule_label> portion of
the source specification has been omitted, so metadata for all schedules as well as the
whole instrument are included.
```

```
>> download dataset_00/sched_CTD/data samplecount=2000 samplestart=0
<< download dataset_00/sched_CTD/data bytecount=56000 samplecount=2000 samplestart=0
<< <The first 2000 samples from the start of the specified schedule><CRC>
    % Here is an alternative start to the transfer of sample data, this time using
samples as the measurement unit.
```

3.5.3 storage

Usage

```
>> storage [ used | remaining | size ]
```

Security

Open.

Description

Reports basic information about the total usage of the data memory. All parameters are read-only:

- **used** is the total number of bytes actually used to store information for all datasets in the memory.
- **remaining** is the number of bytes still available for storage.
- **size** is the maximum total size of the memory in bytes.

For further information about the storage used by each dataset, refer to the **dataset** command.

Some additional notes:

- It is sometimes *not* true that (**used** + **remaining**)=**size**; this is normal. The inherent nature of the physical devices used means that memory must be "allocated" for a particular purpose in blocks that are much larger than the typical sample size. Once such a block has been allocated for one purpose, it can not be used for another, so if some of the block remains unused for its original purpose, it is no longer available.
- The value of **size** is not always exactly the same for all instruments. Some physical devices may have a small percentage of their capacity marked as "bad" by the device manufacturer. These areas are *never* used for information storage, so there is no risk to the user's stored data. The only impact is a slight reduction in usable memory from the nominal capacity.
- When access=usbhost the instrument will not allow dataset downloads through the API. Downloads must be done using the mass storage utility on the host OS.

Examples

```
>> storage
<< storage used=1528 remaining=134216192 size=134217728 access=instrument
```

3.6 Configuration information and calibration

3.6.1 channel

Usage

```
>> channel [ count | list ]
```

```
>> channel <channel_label> [ type | address | settlingtime | readtime | guardtime
| equation | userunits | gain | availablegains | derived | grouplist | sensor ]
```

Security

Unsafe - parameters may not be modified while logging is enabled.

Description

The channels command is used to access information about the channel with the specified <*channel_label*>. Channels are identified by their factory-assigned label, a meaningful string such as temperature_00 or conductivity_00. There are two keys available at the root of the command:

- **count** is the number of channels configured on an instrument
- **list** reports a list of all the channel labels. There is no particular significance to the order in which channels are reported, but for a given instrument the order is fixed. Labels in the list are separated by a pipe character (“|”), with no spaces.

To access information for a specific channel send the <channel_label> as an argument to the channels command. The following parameters give the basic information available for all channels. They are all read-only parameters.

- **type** is a short, pre-defined 'generic' name for the installed channel.
- **address** is the internal address to which this channel responds; it is normally of no interest to end users.
- **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. It applies in a specific situation, namely when the sensor power is turned on, the reading acquired, and then the power is turned off again. In particular, it may not apply in fast sampling modes, when a channel's behaviour may adapt to the need for a higher sample rate.
  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.
- **guardtime** is the minimum time in milliseconds for which the power must remain off once the channel has been powered down. The logger will not turn the channel back on again until this delay has expired.
- **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.
- **factoryunits** 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.
- **derived** is a flag which is either **on** or **off** to indicate whether the channel is a derived channel (**on**) or a measured channel (**off**). This is an intrinsic property of the channel **type**, and can not be modified: it is for information only.

The **derived** parameter is only present in L3.5 and is deprecated. It shall not be used for future backward compatibility.

- **grouplist** reports a list of all the groups that this channel belongs to. The list consists of group labels separated by a vertical bar ('|') character. Users can not directly modify the list using the **channel** command; channels are added to or removed from a group using the **group** command.

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.

- **gain** reports the gain setting currently in use by the channel. 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 <*channel_label*>. 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 does not support multiple gain settings, the response is **none**.
- **sensor** reports the <sensor_label> for a sensor with which the channel is associated. See the **sensor** command for more details.

Examples

```
>> channel
<< channel count=6 list=conductivity_00|temperature_00|pressure_00|backscatter_00|
chlorophyll_00|fdom_00

>> channel conductivity_00
<< channel conductivity_00 type=cond00 address=32 settlingtime=50 readtime=260
guardtime=20 userunits=mS/cm derived=off grouplist=none sensor=none
```

3.6.2 calibration

Usage

```
>> calibration <channel_label> [ equation | datetime | offset | slope | c0 ... cN
| x0 ... xN | n0 ... nN ]
```

Security

Unsafe.

Description

Reports or sets information regarding the most recent calibration for the channel specified by <*channel_label*>, which is a required parameter in all cases (see **channel**). Calibrations cannot be created or deleted. They are tightly coupled to the channels for which they apply and as such the associated <*channel_label*> must be specified to access them. The number and types of coefficients reported, or required when setting, will vary depending on the channel type (see **channel**).

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 labels. 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 by issuing the command without providing any specific 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 (eg. **c3** for a linear sensor) may result in either an error message, or a response such as **c3=na**.

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

- The **equation** and **n**… coefficients are read only.
- **datetime=**<*YYYYMMDDhhmmss*> may accompany any changes to coefficient values, so that the reported date and time reflects the most recent change. If it is not provided when modifying coefficients, the parameter is updated with the current date and time (see the clock command).

Descriptions of the individual parameters are given below.

- **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
- **datetime** is reported and set using a <*YYYYMMDDhhmmss*> format. It is the date and time of the most recent calibration change for the channel.
- **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 can not be changed. They are not coefficients, but (in general) the labels of other logger channels whose data are also inputs to the equation for channel <*channel_label*>. This permits output data to depend on more than one channel; for example, to be corrected for temperature dependencies.
  Most equations which use the **x0**, **x1**… coefficients will require at least one "**n**" entry. 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**" label 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 parameters will be used.
- **offset**, **slope** are provided to permit users to apply a simple linear adjustment to the final value. They are not part of the instrument's official calibration, and RBR Ltd. keeps no record of their values. They can be used, for example, to give a rough correction in the field when a proper re-calibration is not possible. Using these

parameters as a permanent calibration correction is not recommended; many sensors have a non-linear response, or depend also on values from other channels. If a sensor requires recalibration, it should be returned to RBR Ltd for proper handling.

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 voltage_01
<< calibration voltage_01 equation=lin datetime=20171218175005 offset=0.0000000e+000
slope=1.0000000e+000 c0=9.9876543e+000 c1=7.5642301e+000
```

Queries the calibration for a single channel with the label voltage_01.

```
>> calibration voltage_01 c0
<< calibration voltage_02 c0=9.9873456e+000
```

Queries a single parameter for the calibration of a single channel.

```
>> calibration voltage_00 datetime=20171203134201 c0=9.9873456 c1=7.564
<< calibration voltage_00 datetime=20171203134201 c0=9.9873456e+000 c1=7.5640000e+000
```

Setting the calibration for a channel.

3.6.3 sensor

Usage

```
>> sensor [ count | list ]
```

```
>> sensor <sensor_label> [sn | pn | fw | hw | channellist]
```

Security

Open.

Description

A command which returns general sensor information for the instrument.

To access general information about all sensors within an instrument the command can be sent without any arguments. The parameters which will be returned are:

- **count** is the number of sensors installed in the instrument.
- **list** reports a list of all the sensor labels. There is no particular significance to the order in which sensors are reported, but for a given instrument the order is fixed. Labels in the list are separated by a pipe character ('|'), with no spaces.

Parameters are not directly modifiable; they summarize the results of other operations.

```
>> sensor
<< sensor count=3 list=0123456_00|9876543_00|5678901_00
```

To access information for a specific sensor send the <sensor_label> as an argument to the **sensor** command. The following parameters give the basic information available for all sensors. They are all read-only parameters.

- **sn** represents the serial number for the specified sensor.
- **pn** represents the RBR part number for the specified sensor.
- **fw** reports the sensor's current firmware version
- **hw** reports the hardware revision of the sensor
- **channellist** [**=** <*channel_label_list ...*>] reports a list of logger channels that are associated with this sensor. Labels in the list are separated by a pipe character ('|'), with no spaces.

Derived channels will not appear in the **channellist**. Derived channels are not associated with a particular sensor because in most cases they depend on multiple sensors.

```
>> sensor 0123456_00
<< sensor 0123456_00 sn=012345 pn=na fw=na hw=na channellist=conductivity_00|
temperature_00
```

3.6.4 group

Usage

`>> group [ count | maxcount | list ]`

`>> group <group_label> [channellist | schedulelist]`

`>> group create <group_label>`

`>> group delete <group_label> | all`

Security

Open.

Description

This command is used to access information regarding all the channel groups configured within the instrument. Channel groups link instrument channels into a functional group for purpose of sampling. Groups can be created, deleted, accessed, and modified, however, they cannot be renamed. If renaming is required it is recommended to delete the group and create a new group with the appropriate label.

To access meta information about all groups which exist in the instrument, the `group` command can be used without any arguments. The instrument will respond with the following parameters:

- **count** - reports the number of groups currently defined.
- **maxcount** - reports the maximum number of groups that the instrument can hold in its pool at any given time.
- **list** - reports a list of all the defined group labels; labels in the list are separated by a pipe character ("**|**"), with no spaces. Labels are reported in the order that the groups were created, earliest first.

```
>> group
<< group count=3 maxcount=16 list=g.ctd|g.optical|g.chemical
```

To access parameters for a specific group, provide the `<group_label>` as an argument to the `group` command. The following parameters are available for a given group:

- **channellist** reports a list of logger channels that are included in this group. Labels in the list must be separated by a pipe character ("|"), with no spaces. The list must specify at least one channel for the group to be valid. Specifying the keyword **none**, by itself, in place of a list of channel labels, will clear the channel list for the group.
- **schedulelist** is a read-only parameter that reports a list of all the schedules currently in the pool which use this group. Any group usage for historical datasets stored in the logger's memory is not included. Labels in the list are separated by a pipe character ("|"), with no spaces. If the group is unused by any schedules, this list is replaced by the word **none**.

```
>> group salinity_grp
<< group salinity_grp channellist=conductivity_00 schedulelist=none
```

If only a single parameter is needed, it can be requested by providing the key for that parameter as an argument to the group specified by the `<group_label>`.

```
>> group salinity_grp schedulelist
<< group salinity_grp schedulelist=none

>> group salinity_grp channellist
<< group salinity_grp channellist=conductivity_00
```

Modifications can only be performed on a single group at a time. To perform a modification, provide the `<group_label>` as an argument to the `group` command then provide the key/value pairing for the value to be changed. There is only one available key which can be modified and it is:

- **channellist** [**=**<*channel_label_list ...*>] sets a list of logger channels that are included in this group. Labels in the list must be separated by a pipe character ("|"), with no spaces. The list must specify at least one channel for the group to be valid. Specifying the keyword **none**, by itself, in place of a list of channel labels, will clear the channel list for the group.

```
>> group salinity_grp channellist=conductivity_00|temperature_00|pressure_00
<< group salinity_grp channellist=conductivity_00|temperature_00|pressure_00

>> group salinity_grp channellist=salinity_00|conductivity_00|temperature_00|
pressure_00|temperature_01
<< group salinity_grp channellist=salinity_00|conductivity_00|temperature_00|
pressure_00|temperature_01
```

3.6.4.1 create

Usage

`>> group create <group_label>`

Security

Unsafe - groups may not be created while logging is enabled.

Description

Groups can be created the **create** action within the **group** command. The group label must be specified as an argument to the action.

```
>> group create g.pressure
<< group create g.pressure

>> group
<< group count=4 maxcount=16 list=g.ctd|g.optical|g.chemical|g.pressure
```

3.6.4.2 delete

Usage

`>> group delete <group_label> | all`

Security

Unsafe - groups may not be deleted while logging is enabled.

Description

Deletes one or all groups. Groups are specified by their labels. At least one existing group must be specified. The parameters are as follows:

- <*group_label*> specifies by label a single group to delete from the pool.
- **all** causes all user-defined groups to be deleted from the pool.

A deleted group can no longer be used by any schedule. Any schedule in the pool of schedules which used the deleted group will automatically have the group removed from its **list**. Historical datasets in the logger memory are not affected when a group is deleted; the configuration snapshot saved in the dataset's metadata includes all group details at the time the logger was enabled.

```
>> group
<< group count=4 maxcount=16 list=g.ctd|g.optical|g.chemical|g.pressure

>> group delete g.optical
<< group delete g.optical

>> group
<< group count=3 maxcount=16 list=g.ctd|g.chemical|g.pressure
```

Deleting `g.optical` removes it from the pool and reduces the count by one.

```
>> group
<< group count=3 maxcount=16 list=g.ctd|g.chemical|g.pressure

>> group delete all
<< group delete all

>> group
```

```
<< group count=0 maxcount=16 list=none
```

Deleting `all` groups removes all items from the pool and reduces the count to zero. The schedule `list` will indicate it is empty with the value of `none`.

3.6.5 schedule

Usage

`>> schedule [ count maxcount list availablemodes availablefastperiods]`

`>> schedule <schedule_label> [ grouplist | configlist | stream | storage | mode | <mode_dependent_parameters>]`

`>> schedule create <schedule_label>`

`>> schedule delete <schedule_label> | all`

Security

Open.

Description

The `schedule` command can be used to create, delete, modify, and access sampling schedules within the instrument.

To access meta information about all the configured schedules in the instrument, the `schedule` command can be specified without any arguments. If a specific parameter is required, the parameter key can be provided as an argument to the `schedule` command. The list of parameter keys are as follows:

- **count** reports the number of schedules currently defined.
- **maxcount** reports the maximum number of schedules that the logger can hold in its pool at any given time.
- **list** reports a list, by label, all defined schedules; labels in the list are separated by a pipe character ("|"), with no spaces. Labels are reported in the order that the schedules were created, earliest first.
- **availablemodes** lists all the sampling modes configured to be available in the instrument; not all instruments support all modes. Items in the list are separated by a pipe character ("|"), with no spaces.
- **availablefastperiods** reports a list of the fast measurement periods available to the instrument 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, "|". The **period** for sampling rates faster than 1Hz can be set to any value in the list, but no others. If there are no fast periods available, the word **none** is returned instead of a list of values. Note that the periods given in the list may be supported in some modes but not others, depending on the instrument's configuration.

```
>> schedule
<< schedule count=3 maxcount=16 list=test|schedule_04|basic_schedule
availablemodes=continuous|average|tide availablefastperiods=500|250|125|63
```

In order to access or modify properties for a specific schedule, provide the `<schedule_label>` as an argument to the `schedule` command. The following parameters are available for a given schedule:

- <*schedule_label*> is a required parameter that specifies by label the schedule to access. Labels of schedules cannot be renamed.
- **grouplist** [**=**<*group_label_list ...*>] reports or sets a list of groups defining the channels that will be sampled according to this schedule. Labels in the list must be separated by a pipe character ("|"), with no spaces. The list must specify at least one group for the schedule to be valid. Specifying the keyword **none**, by itself, in place of a list of group labels, will clear the group list.
- **configlist** [**=**<*config_label_list ...*>] is a read-only parameter that reports a list of all the configurations currently in the pool which use this schedule. Any configurations used for historical datasets stored in the logger's memory are not included. Labels in the list are separated by a pipe character ("|"), with no spaces. If the schedule is unused by any configurations, this list is replaced by the word none.
- **stream** [**=serial** | **usb** | **off**] is used to report or set the communication link over which data for this schedule will be streamed in real time during the deployment. At most one link may be active for each schedule. Defaults to **off** when the schedule is created.
- **storage** [**=on** | **off**] determines whether data for this schedule will be stored in memory during the deployment. For data logging instruments which can store data, this parameter defaults to **on** when the schedule is created. For sensor instruments that do not store data, schedules are created with this parameter set to **off**, and the value can not be changed.
- **mode** [**= continuous** | **regimes**] reports or sets the sampling mode to be used by this schedule.
- <*mode_dependent_parameters*> will be described in more detail below.

Here are some points to be aware of when modifying a schedule.

- Any modification made to a schedule will cascade into all configurations currently in the pool which use the schedule. Historical datasets in the logger's memory are not affected.
- The order of <*group_labels*> in the <*group_label_list*> determines the order in which channels will be sent in real time and stored in memory for this schedule. For example, if **group_A** contains channels W and X, and **group_B** contains channels Y and Z, then a group list of **group_A|group_B** results in a channel order **W,X,Y,Z**, whereas **group_B|group_A** would result in **Y,Z,W,X**.
- It is possible to specify a <*group_label*> for a group that does not yet exist, but at some point the group must be defined before the schedule can be used in a deployment.

Mode dependent parameters in each schedule are reported, and can be modified, only for the sampling mode currently selected. Parameters for only one mode at a time can be held in non-volatile memory for each schedule; if the mode for a schedule is changed, settings for the old mode are lost, and settings for the new mode start with a pre-defined set of default values. The mode and settings of other schedules are not affected.

In all cases there may be instrument specific constraints on the parameter values that can be used, but here are some general guidelines:

- A period must either be a multiple of 1000ms, or be specified from the list of **availablefastperiods** (see the **schedules** command). The upper limit is 86400000ms, corresponding to 24 hours. The period values reported in **availablefastperiods** all correspond to exact frequencies in Hz, rounded to the nearest millisecond; for example, 125ms for 8Hz, 63ms for 16Hz. See also Tips for system integrators.

continuous mode

A simple mode in which data is acquired at a single, steady rate between the start and the end of the deployment. The parameters are as follows:

- **period** [**=** <*milliseconds*>], the amount of time between consecutive samples; must comply with the generic constraints on sampling periods given above.
- **castdetection**=**off** is reported for consistency with other types of instrument. It can not be set to **on**, and attempting to do so will provoke an error message.

regimes

This mode is intended for use when the instrument is integrated into a moving platform. The water column is divided into regions by depth, with a maximum of three regions, or **regimes**. Each regime can use a different sampling rate, and within each regime the water column is divided into bins, which are again defined by depth. All data acquired in each bin is averaged before being stored. For a more detailed discussion of the regimes sampling mode, refer to the chapter Integrating with a profiling float in the Quick start section. The parameters for **regimes** mode are as follows:

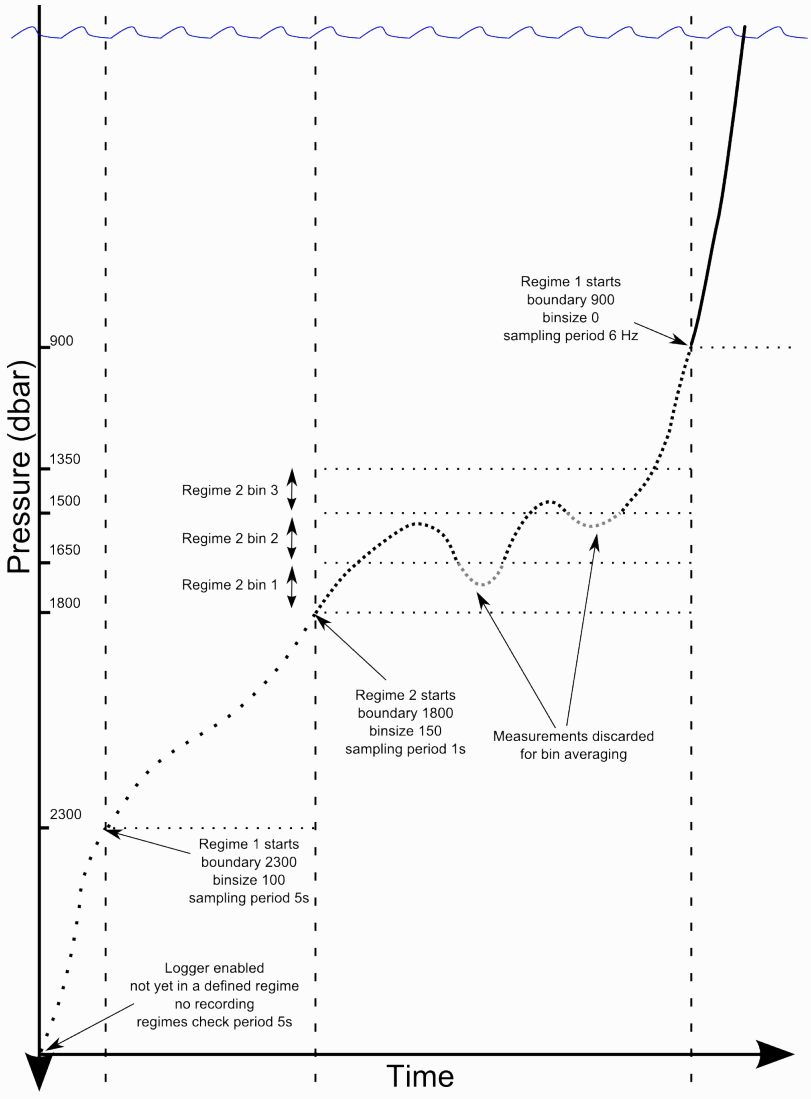
- **direction** [**= ascending** | **descending**]
  This parameter indicates the intended vertical direction of the instrument through the water column while sampling. **ascending** means that sampling will start at depth and continue until the instrument reaches the surface, whereas **descending** means that sampling starts at (or near) the surface and continues as the instrument depth increases.
- **count** [**= 1** | **2** | **3**]
  This parameter indicates the number of regimes that are set; the minimum is 1 and the maximum is 3. When multiple regimes are used, regime 1 is always where sampling begins; that is, if **direction**=**ascending** then regime 1 is the deepest, whereas if **direction**=**descending** then regime 1 is the shallowest.
- **reference** [=<**channel label>**]
  This parameter indicates which channel is used to determine which regime, and which bin within that regime, currently applies during sampling. This channel can either be an absolute or sea pressure channel. Sea pressure is the difference between the absolute pressure measured and the atmospheric pressure. Most instruments do not independently measure atmospheric pressure; in this case a nominal fixed value is defined via the **settings** command.
- **finalboundary** [=<*dbar*>]
  This parameter specifies the transition depth to cease sampling. So, if **direction**=**ascending**, the boundary value is the *uppermost* depth of all of the regimes, whereas if **direction**=**descending**, the boundary value is the *lowest* depth of any enabled regime. The value is given in units of dbar to a precision of 1 dbar; fractional dbar values are not accepted. Before the instrument can be deployed, the **verify** and **enable** commands will check that all regimes boundaries are strictly increasing if **direction**=**descending**, or strictly decreasing if **direction**=**ascending**, and respond with an error message if the constraint is violated.
- **boundary1** [=<*dbar*>]
  This parameter specifies the transition depth from one regime to the next. The boundary value specified for each individual regime is the depth at which sampling starts in that regime. So, if **direction**=**ascending**, the boundary value is the *lowest* depth of the regime, whereas if **direction**=**descending**, the boundary value is the *uppermost* depth of the regime. The value is given in units of dbar to a precision of 1 dbar; fractional dbar values are not accepted. Before the instrument can be deployed, the **verify** and **enable** commands will check that all regimes boundaries are strictly increasing if **direction**=**descending**, or strictly decreasing if **direction**=**ascending**, and respond with an error message if the constraint is violated.
- **binsize1** [=<*dbar*>]
  This parameter specifies the size (depth range) used for each averaged bin. It is typically set by the user so that the total regime size is an integer multiple of the bin size; however, if the last bin in the regime is smaller than the rest, the measurement is stored "early" and the next regime commences. The bin size is specified in units of dbar, with a precision of 0.1 dbar. If the bin size is set to 0.0, the logger will not average the data per bin during the regime and will just record/stream every measurement.
- **period1** [=<*milliseconds*>]
  This parameter specifies the amount of time between consecutive samples throughout all bins in the given regime. The value must comply with the generic constraints on sampling periods given above.
- **boundary2** [=<*dbar*>]
- **binsize2** [=<*dbar*>]

- **period2** [=<*milliseconds*>]
- **boundary3** [=<*dbar*>]
- **binsize3** [=<*dbar*>]
- **period3** [=<*milliseconds*>]
  Parameters for the second and third regimes, if used, are exactly analogous to those for the first regime already described.

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, travelling in the specified **direction**. For example, in the following figure the average of the first bin in regime 2 is stored as soon as the vehicle enters the second bin. When the vehicle goes back into the range defined for the first bin, these measurements are discarded, but all measurements acquired in the second bin continue to be accumulated, as long as the vehicle has not entered yet the third bin.

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



count channel

The logger may be configured with a channel of type **cnt_00**. This is a derived channel designed to count the number of readings that actually contributed to an averaged value; it may not be included in every logger configuration.

It is most useful in **regimes** sampling mode, where it will indicate the number of samples averaged to obtain the reading in each bin. It may also be used in the **average** or **tide** sampling modes, where it will usually have the same value as the **measurementcount** parameter, unless it was necessary to omit some readings from the average because of errors. For other sampling modes, the value of this channel (if present) will just be 1; remember that the channel can be excluded from any or all groups if it is not required.

In all cases, if the channel is sampled according to multiple schedules, the value reported for each schedule could be different and, in each case, will be applicable to the schedule using it.

3.6.5.1 create

Usage

```
>> schedule create <schedule_label>
```

Security

Unsafe - schedules may not be created while logging is enabled.

Description

Schedules can be created and deleted using the **create** or **delete** actions within the **schedules** command. In both cases <schedule_label> must be specified as an argument to the action.

```
>> schedule
<< schedule count=3 maxcount=16 schedulelist=test|schedule_04|basic_schedule
availablemodes=continuous|average|tide,availablefastperiods=500|250|125|63

>> schedule create s.pressure
<< schedule create s.pressure

>> schedule s.pressure
<< schedule s.pressure grouplist=none configlist=none stream=off storage=on
mode=continuous period=1000 castdetection=off

>> schedule
<< schedule count=4 maxcount=16 schedulelist=test|schedule_04|basic_schedule|s.pressure
availablemodes=continuous|average|tide availablefastperiods=500|250|125|63
```

3.6.5.2 delete

Usage

```
>> schedule delete <schedule_label> | all
```

Security

Unsafe - schedules may not be deleted while logging is enabled.

Description

Deletes one or all schedules. Schedules are specified by their labels. At least one existing schedule must be specified. The parameters are as follows:

- <*schedule_label*> specifies by label a single schedule to delete from the pool.
- **all** causes all user-defined schedules to be deleted from the pool.

A deleted schedule can no longer be used in any configuration. Any configuration in the configuration pool which used the deleted schedule will automatically have the schedule removed from its **list**. Historical datasets in the logger memory are not affected when a schedule is deleted; the configuration snapshot saved in the dataset's metadata includes all schedule details at the time the logger was enabled.

Schedules can be deleted using the **delete** action within the **schedule** command. The `<schedule_label>` must be specified as an argument to the action.

```
>> schedule
<< schedule count=4 maxcount=16 list=test|schedule_04|basic_schedule|s.pressure
availablemodes=continuous|average|tide availablefastperiods=500|250|125|63

>> schedule delete schedule_04
<< schedule delete schedule_04

>> schedule
<< schedule count=3 maxcount=16 list=test|basic_schedule|s.pressure
availablemodes=continuous|average|tide availablefastperiods=500|250|125|63
```

Deleting `schedule_04` removes it from the pool and reduces the count by one.

```
>> schedule
<< schedule count=3 maxcount=16 list=test|basic_schedule|s.pressure
availablemodes=continuous|average|tide availablefastperiods=500|250|125|63

>> schedule delete all
<< schedule delete all

>> schedule
<< schedule count=0 maxcount=16 list=none availablemodes=continuous|average|tide
availablefastperiods=500|250|125|63
```

Deleting `all` schedules removes all items from the pool and reduces the count to zero. The schedule `list` will indicate it is empty with the value of `none`.

3.6.6 config

Usage

```
>> config [ count maxcount list ]
```

```
>> config <config_label> [ schedulelist ]
```

```
>> config create <configuration_label>
```

```
>> config delete <configuration_label> | all
```

Security

Open.

Description

This command is used to access information regarding all of the sampling configurations within the instrument. Configurations contain a list of schedules to be run in a deployment. The configuration is specified at the time of enabling the instrument. Configurations can be created, deleted, accessed, and modified, however, they cannot be renamed. If renaming is required it is recommended to delete the configuration and create a new configuration with the appropriate label.

To access meta information about all configurations which exist in the instrument, the `config` command can be used without any arguments. If a specific parameter is required, the parameter name can be provided as an argument to the config command. In this case only that parameter will be returned in response. The list of parameters are as follows:

- **count** reports the number of configurations currently defined.
- **maxcount** reports the maximum number of configurations that the logger can hold in its pool at any given time.
- **list** reports by label all defined configurations; labels in the list are separated by a pipe character ("**|**"), with no spaces. Labels are reported in the order that the configurations were created, earliest first.

```
>> config
<< config count=5 maxcount=16 list=test_config|config_01|config_02|cfg_profiling|
default_config
```

```
>> config list
<< config list=test_config|config_01|config_02|cfg_profiling|default_config
```

To access parameters for a specific config, provide the `<config_label>` as an argument to the `config` command. The following parameters are available for a given config:

- **schedulelist** reports a list of schedules to be executed when this configuration is used to enable a deployment. Labels in the list must be separated by a pipe character ('|'), with no spaces. The list must specify at least one valid schedule before the configuration can be used. Specifying the keyword **none**, by itself, in place of a list of schedule labels, will clear the schedule list for the config. A maximum of **eight** schedule labels can be added to the schedulelist.

```
>> config
<< config count=5 maxcount=16 list=test_config|config_01|config_02|cfg_profiling|
default_config

>> config config_01
<< config config_01 schedulelist=default_schedule
```

Modifications can only be performed on a single config at a time. In order to access or modify properties for a specific config, provide the `<config_label>` as an argument to the `configs` command. Currently there is only a single parameter available for a given config:

- **schedulelist** reports or sets a list of schedules to be executed when this configuration is used to enable a deployment. Labels in the list must be separated by a pipe character ("|"), with no spaces. The list must specify at least one valid schedule before the configuration can be used. Specifying the keyword **none**, by itself, in place of a list of schedule labels, will clear the schedule list for the config.

A maximum of **eight** schedule labels can be added to the schedulelist.

Here are some points to be aware of when modifying a configuration.

- Any modification made to a configuration will apply only when it is used for future deployments; historical datasets in the logger's memory are not affected.
- The order of <*schedule_labels*> in the <*schedule_label_list*> does not matter.

```
>> config cfgPrimary
<< config cfgPrimary schedulelist=default_schedule

>> config cfgPrimary schedulelist = schedule_fast|schedule_burst
<< config cfgPrimary schedulelist=schedule_fast|schedule_burst

>> config cfgPrimary
<< config cfgPrimary schedulelist=schedule_fast|schedule_burst
```

3.6.6.1 create

Usage

`>> config create <configuration_label>`

Security

Unsafe - configurations may not be created while logging is enabled.

Description

A configuration can be created using the **create** actions within the **config** command. The `<config_label>` must be specified as an argument to the action.

```
>> config
<< config count=5 maxcount=16 list=test_config|config_01|config_02|cfg_profiling|
default_config
```

```
>> config create standard_config
<< config create standard_config

>> config
<< config count=6 maxcount=16 list=test_config|config_01|config_02|cfg_profiling|
default_config|standard_config
```

3.6.6.2 delete

Usage

`>> config delete <configuration_label> | all`

Security

Unsafe - configurations may not be deleted while logging is enabled.

Description

Deletes one or all configurations. Configurations are specified by their labels. At least one existing configuration must be specified. The parameters are as follows:

- <*configuration_label*> specifies, by label, a single configuration to delete from the pool.
- **all** causes all user-defined configurations to be deleted from the pool.

A deleted configuration can no longer be used for a future deployment. Any historical deployments in the logger memory that used the deleted configuration are not affected; all datasets include as part of their metadata a snapshot of the configuration when the logger was enabled.

```
>> config
<< configs count=6 maxcount=16 list=test_config|config_01|config_02|cfg_profiling|
default_config|standard_config

>> config delete config_01
<< config delete config_01

>> config
<< config count=5 maxcount=16 list=test_config|config_02|cfg_profiling|default_config|
standard_config
```

Deleting `config_01` removes it from the pool and reduces the count by one.

```
>> config
<< configs count=6 maxcount=16 list=test_config|config_01|config_02|cfg_profiling|
default_config|standard_config

>> config delete all
<< config delete all

>> config
<< config count=0 maxcount=16 list=none
```

Deleting `all` configurations removes all items from the pool and reduces the count to zero. The configuration `list` will indicate it is empty with the value of `none`.

3.6.7 settings

Usage

```
>> settings [ prompt [=on|off]] [confirmation [=on|off]]
```

Security

Open.

Description

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

- **prompt** specifies whether the instrument returns the “Ready:” prompt following a response. The as-shipped default value is **on**.
- **confirmation** specifies whether the instrument returns a response from a create or set/modify operation to verify the new state. The as-shipped default value is **on**.
  A response will always be sent when a parameter value is simply requested.

Examples

```
>> settings
<< settings prompt=on confirmation=on
```

Request all settings

```
>> settings confirmation
<< settings confirmation=on
```

Request only the **confirmation** setting

```
>> settings prompt=off
<< settings prompt=off
```

Update the **prompt** setting

3.6.8 parameters

Usage

```
>> parameters [ altitude | atmosphere | avgsoundspeed | density | pressure |
salinity | speccondtempco | temperature ]
```

Security

Unsafe.

Description

Reports or sets channel parameters which may be required when computing calibrated output in the logger as described below.

- **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 instrument, and if wave analysis is not required, the parameter can be ignored.
- **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.
- **atmosphere**, **avgsoundspeed**, **density**, **pressure, salinity**, **temperature** [=<*value*>]: these are default parameter values, to be used when the instrument 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.

Parameter	Units	Default value
altitude	m	0
atmosphere	dbar	10.132501
avgsoundspeed	m/s	1506.8
density	g/cm^3	1.026021
pressure	dbar	10.132501
salinity	PSU	35
speccondtempco		0.0191
temperature	°C	15.0

Examples

```
>> parameters atmosphere
<< parameters atmosphere=10.132501
```

Request only the atmosphere parameter

```
>> parameters density=1.0295
<< parameters density=1.0295
```

Update the density parameter

```
>> parameters
<< parameters altitude=0.0000 atmosphere=10.1325010 avgsoundspeed=1506.8000
density=1.0260206 pressure=10.1325010 salinity=35.0000 speccondtempco=0.0191
temperature=15.0000
```

Request all parameters

4 Format of stored data

4.1 Overview

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

Three major types of deployment information are stored in a instrument's memory:

- Sample data, comprising sets of measured values from all active channels in the instrument.
- Events, which are records of non-sample incidents used to aid interpretation, or for diagnostics.
- Metadata, which contains meta-information about the instrument and the deployment parameters used for each schedule.

All types of data are contained in "storage objects"; a storage object may be thought of as a file. Depending on the type of memory used by the instrument, this may actually be a file in a folder that is part of a larger file system. However, not all instruments implement memory in this way, and may not allow all the common file operations, but thinking of the storage object as a file is still a useful convenience for notation.

4.1.1 Sample data

Sample data is stored according to the schedules executed by the configuration when acquiring the data for the deployment; there is one storage object for each schedule. Only data from the channels associated with a schedule will appear in its storage object; data from other channels will be found in the storage objects for other schedules. However, it is possible for any channel to be associated with more than one schedule, in which case its data will appear in each of the associated storage objects. A channel that is not associated with *any* schedule that was executed by this configuration is considered to be inactive; it will not have been sampled, and so will have no data in the memory.

The format of the sample data within a storage object for a schedule is detailed further in the Section Sample data storage format that follows.

4.1.2 Events

Events are typically all contained in a single storage object. Some event types apply to the entire instrument, others may apply to a single schedule, while yet others may apply only to a single channel. For example:

- An event indicating a change from external power to internal batteries applies to the whole instrument.
- A cast detection event would apply to a schedule.
- An event indicating a change of sensor gain would apply to only a single channel.

This information is coded within the event itself, so that events may be filtered according to different criteria when retrieved.

The format of the event data within the storage object is detailed further in the Section Event data storage format that follows.

4.1.3 Metadata

Metadata is typically all contained in a single storage object, and includes all the information necessary to describe the state of the instrument when the deployment was enabled. This information falls into several categories, or levels, and overview of which is:

- Details of the instrument that do not change over time (for example, its serial number).
- Settings that apply to the whole instrument and may be changed, although once set they are modified rarely, if ever (for example, the serial baud rate).
- Settings that apply to the whole instrument and may commonly be changed from one deployment to the next (for example, the start time).

- The organization of schedules, groups and channels used by the configuration for this deployment.
- The groups and sampling parameters used by each schedule.
- The membership of channels within groups.
- Details of each channel used, including calibration coefficients and any other specific information that applies.

An attempt is made to limit the metadata to information that is relevant to the deployment. Information about any channel, group or schedule may typically *not* appear in the metadata if the item was not used by the configuration for this deployment, even though the item is defined in the instrument.

The structure of the metadata can be quite complex, depending on the number of schedules and groups involved, but it is organized so that metadata can be filtered according to what is applicable to any particular schedule.

Not all users will need or want access to the metadata, but for those who do, the organization and format is detailed further in the Section Metadata layout that follows.

4.1.4 Related commands

dataset, an overview of all of the datasets stored in the instrument as well as details for an individual dataset.

download, retrieve all or part of the information for a dataset.

enable, create a dataset when the instrument is enabled for deployment.

storage, obtain general information about the instrument's data storage.

4.2 Sample data storage format

4.2.1 Options

float32	Each reading is stored in IEEE single precision 32-bit floating point format.
float64	Each reading is stored in IEEE double precision 64-bit floating point format.
calfloat64	Numerically the same as **float64**, but no calibration equation is applied to the data; the value is a ratio relative to full-scale, nominally 0.0 to 1.0.

The format to expect for all schedules can be determined by the **storage** command (see Related commands below) or from the metadata entry **data_format** in the Deployment section (section 4) of the metadata. Once the value for *normal* deployments is known for a given instrument, it can be assumed that it will not change during the instrument's lifetime. The only case when the format may be different is if the deployment was enabled in calibration mode, in which case the format will be **calfloat64**.

4.2.2 Layout

Within the sample data for a given schedule, a single sample with N channels (N > 0) will have the form:

Timestamp	1st channel value	**...**	Nth channel value

All individual data items are stored in memory in little-endian format (least significant byte at lower address). Within a given schedule, all samples occupy the same number of bytes each.

The **Timestamp** is a 64-bit count of milliseconds elapsed since 1970/01/01 00:00:00, commonly referred to as the Unix epoch. It accounts for leap years, but *not* leap seconds, time zones or any other adjustments. Note that although the

Unix epoch is used as a reference point, the time does not conform exactly to the definition of "Unix time", which is measured in seconds.

All **channel values** have the same numeric format; one of **float32**, **float64** or **calfloat64** (see Options, above). The order in which channel values appear is determined by:

1. The order in which groups are specified for this schedule.
2. Within each group, the order in which channels are specified.

These orderings are determined by the **schedule** and **group** commands (see Related commands, below).

All **channel values** have the potential to encode an error value. Error values are denoted by NaNs with an embedded payload (see Errors, below)

4.2.3 Related commands

instrument

>> **instrument datatype**
<< **instrument datatype=float32** | **float64** | **calfloat64**

- Read-only
- Sample data is in the same format for all channels.
- Normal deployments will be in either **float32** or **float64**, the choice is configured at the Factory.

enable

>> **enable config=**<*config_label*> **dataset=**<*dataset_label*> **storagemode=normal | calibration**
<< **enable config=**<*config_label*> **dataset=**<*dataset_label*> **storagemode=normal | calibration**

- **storagemode** is optional and defaults to **normal** if not specified.
- **normal** selects a data storage format of either **float32** or **float64**, according to the logger's configuration.
- **calibration** always selects a data storage format of **calfloat64**, regardless of the **normal** setting.

schedule

>> **schedule grouplist=**<*group_label_list ...*>
<< **schedule grouplist=**<*group_label_list ...*>

- The order in which group labels are specified determines the order in which groups appear in the stored data.

group

>> **group channellist=**<*channel_label_list ...*>
<< **group channellist=**<*channel_label_list ...*>

- The order in which channel labels are specified determines the order in which channels appear within their group in the stored data.

dataset

>> **dataset** <*dataset_label*> **datatype**
<< **dataset** <*dataset_label*> **datatype=float32** | **float64** | **calfloat64**

- The data storage format of a historical dataset held in the logger's memory can be queried at any time.

4.2.4 Errors

Errors related to channels are denoted by negative-signed NaN values. In many cases, detecting a NaN as a channel value should suffice to indicate an error has occurred since most error codes are diagnostic/correctable for RBR-only.

For conversion from **float32** to **float64**, a 29-bit least-significant-bit-zero-pad of the payload is used (mantissa is left-shifted 29 bits). Conversely, for **float64** to **float32**, a 29-bit least-significant-bit-truncation of the payload is used (mantissa is right-shifted 29 bits). This conversion matches several C compiler and Java intrinsic conversions, but integrators should check intrinsic conversions in their own environments. Except for the case of a general error, the setting of the quiet/signalling bit (**float32**: 0x00400000, **float64**: 0x0008000000000000) should **not** be relied on for error determination; for a general error, the bit will be set.

Error	float32 bits	float64 bits	Description
	0x7F800000	0x7FF0000000000000	+inf
	0xFF800000	0xFFF0000000000000	-inf
	0x7FC0000	0x7FF0000000000000	+NaN
0	0xFFC0000	0xFFF8000000000000	-NaN; General error condition; error from undefined mathematical operation
1	0xFFC00001	0xFFF8000020000000	ADC error – end of conversion
2	0xFFC00002	0xFFF8000040000000	ADC error – invalid value
3	0xFFC00003	0xFFF8000060000000	Bus error – invalid address
4	0xFFC00004	0xFFF8000080000000	Bus error – frame overflow
5	0xFFC00005	0xFFF80000A0000000	Bus error – locked
6	0xFFC00006	0xFFF80000C0000000	Bus error – cannot transmit
7	0xFFC00007	0xFFF80000E0000000	Bus error – receive timed out
8	0xFFC00008	0xFFF8000100000000	Bus error – invalid frame
9	0xFFC00009	0xFFF8000120000000	Sample error – no sample started
10	0xFFC0000A	0xFFF8000140000000	Sample error – sample in progress
11	0xFFC0000B	0xFFF8000160000000	Sample error – sample failed
12	0xFFC0000C	0xFFF8000180000000	Sample error – averaging failed
13	0xFFC0000D	0xFFF80001A0000000	Bus error – packet truncated
14	0xFFC0000E	0xFFF80001C0000000	Data error – unable to compute
15	0xFFC0000F	0xFFF80001E0000000	Safety – high power consumption

Error	float32 bits	float64 bits	Description
16	0xFFC00010	0xFFF8000200000000	Data error – out of range
17	0xFFC00011	0xFFF8000220000000	Data error – under range
18	0xFFC00012	0xFFF8000240000000	Data error – over range
19	0xFFC00013	0xFFF8000260000000	Sensor error – communications timeout
20	0xFFC00014	0xFFF8000280000000	Sensor error – cannot parse response
21	0xFFC00015	0xFFF80002A0000000	Data error – not calibrated / invalid calibration
22	0xFFC00016	0xFFF80002C0000000	Data error – malformed floating point number
23	0xFFC00017	0xFFF80002E0000000	Data error – no sample logged

4.3 Metadata layout

- All multi-byte data items are stored in memory in little-endian format (least significant byte at lower address).
- Except for Section 1, it should not be assumed that all sections are present, or that they are in order.

Section	ID	Description
TAG		A special tag. Ruskin or external reader can use it to check if the data is a L3.5/Gen4 header.
Metadata	1	A high level map of the metadata, used for navigation, not required to interpret the metadata itself.
Logger	2	Fixed information about the logger.
Settings	3	User settings that do not directly impact a deployment, and may not even be changed very often between deployments.
Deployment	4	User settings that do impact the values, storage, or presentation of deployment data for all schedules, or which are otherwise closely related to the deployment.
Configuration	5	Top level information about the configuration used for the deployment.
Schedules	6	Organized to allow extraction of individual schedules to construct metadata on a per-schedule basis.
	6.1	Schedule map. Only those schedules used by this configuration appear in the map.

Section	ID	Description
	6.2.1	Schedule details. Details of the 1st schedule.
	6.2.2	Schedule details. Details of the 2nd schedule. Details as above for 1st schedule.
	...	...
	6.2.S	Schedule details. Details of the S th schedule. Details as above for 1st schedule.
User-groups	7	Organized to allow extraction of individual groups to construct metadata on a per-schedule basis.
	7.1	Group map. Only those groups used by this configuration appear in the map.
	7.2.1	Group details. Details of the 1st user-group.
	7.2.2	Group details. Details of the 2nd user-group. Details as above for 1st group.
	...	...
	7.2.G	Group details. Details of the G th user-group. Details as above for 1st group.
Module-groups	8	If present, used for diagnostic purposes.
	8.1	Module group map.
	8.2.1	Module group details. Details of the 1st FE-group.
	8.2.2	Module group details. Details of the 2nd FE-group. Details as above for 1st FE-group.
	...	...
	8.2.F	Module group details. Details of the F th FE-group. Details as above for 1st FE-group.
Channels	9	Internal map, then each channel has its own self-contained subsection, so that individual channels can be extracted to construct metadata on a per-schedule basis.
	9.1	Channel map outlining where the channel details can be found within the following sections.

Section	ID	Description
	9.2.1	Channel details. Details of the 1st channel.
	9.2.2	Channel details. Details of the 2nd channel. Details as above for 1st channel.
	...	...
	9.2.C	Channel details. Details of the C th channel. Details as above for 1st channel.

4.3.1 TAG

A special tag. Ruskin or external reader can use it to check if the data is a L3.5/Gen4 header.

Item	Size (bytes)	Comment
0x00524252	4	A tag to indicate L3.5/Gen4 type header ("RBR\0"). L3 deployment header start with byte 0x01.

4.3.2 Metadata

A high level map of the metadata, used for navigation, not required to interpret the metadata itself.

Item	Size (bytes)	Comment
Section ID	4	0x01000000, "1.0.0.0"
Section size	2	Variable; omitted sections have no map entry. Includes section CRC.

Metadata version	4	Major.Minor.Patch, eg. 0x01166AA5, "1.22.27301"
		
		`0x00020000, "00.02.0000" - Added hash code`
		`0x00010000, "00.01.0000" - Initial release`

Total size	4	Complete size of all metadata, including all CRCs.
Hash code	4	Unique deployment identifier. `Introduction: v00.02.0000`

Section ID (32b)	Offset (32b)	Size (16b)	Comment
0x01000000	0x00000004	variable	Section 1 always present at offset 4.
0x02000000	variable	variable	Includes any and all 2.X.X.X sections.
....	....	variable	Except for Section 1, it should not be assumed that all sections are present, or that they are in order.
0x08000000	variable	variable	Includes any and all 8.X.X.X sections. Section 8 is the last in this example.

Followed by a standard 16b CRC of all Section 1.

4.3.3 Logger

Fixed information about the logger.

Item	Size(bytes)	Comment
Section ID	4	0x02000000, "2.0.0.0"
Section size	2	Variable; part number sizes may vary. Includes CRC.
f/w type	4	120 for L3.5, 130 for SL4
fw_version_string	36	NUL-terminated firmware version string, maximum 35 printable characters. Example for Gen4, including semantic version number, special identifiers, and build date/time: `0.1.1-dev+202307281615`
serial_number	4	
model_name	16	Max 15 printable characters, NUL terminated, 0xFF-padded if necessary.
permissions	4	32b flags for *permitted* features. Bit flags for feature permissions.
cell_count	2	Battery capacity in AA cells, usually 0, 4, or 8.
cell_format [1]	16	Max 15 printable characters, NUL terminated, 0xFF-padded if necessary.
fe_baudrate	4	Baud rate on FE-bus.

Item	Size(bytes)	Comment
pn_size	2	Length of part number, includes terminating NUL.
part_number	variable	Part number string, does not include a NUL terminator.
psu_pn_size	2	Length of PSU part number, includes terminating NUL.
psu_part_number	variable	Power supply part number string, does not include a NUL terminator.
CRC	2	A standard 16b CRC of all Section 2.

[1] An arbitrary string that the logger does not *currently* use internally.

4.3.4 Settings

User settings that do not directly impact a deployment, and may not even be changed very often between deployments.

Item	**Size (bytes)**	**Comment**
Section ID	4	0x03000000, "3.0.0.0"
Section size	2	As things stand, 44. Includes CRC.
serial_baudrate	4	The baud rate as an integer number.
serial_mode	4	Enumerated code. See **Serial mode definitions.**
RESERVED	4	
RESERVED	4	
RESERVED	4	
wifi_initial_timeout	4	seconds - CHANGED to milliseconds for consistency.
wifi_command_timeout	4	seconds - CHANGED to milliseconds for consistency.
poll_poweroff_delay	4	milliseconds - formerly **fetch.**

Item	Size (bytes)	Comment
feature_usage	4	Bit flags showing usage of instrument-wide features; moved here from the Deployment section. See **Feature flag bit assignments**.
CRC	2	A standard 16b CRC of all Section 3.

4.3.4.1 Feature flag bit assignments

Flag	Bit	Comment	1	0
PROMPT	b00	prompt was turned on/off	on	off
CONFIRMATION	b01	confirmation was turned on/off	on	off
FWLOCK	b10	f/w updates locked/permitted	locked	permitted
WETSWITCH	b14	indicates whether the wetswitch gate is used or not	used	unused
SENSORPOWERALWAYSON	b15	sensors kept powered up between samples, or not	always on	normal
TWISTACTIVATION	b16	indicates whether the twist activation gate used or not	used	unused
RESERVED	b19		used	unused
SIMULATED_DATA	b20	simulated data in use or not	simulated	normal
WIFI	b22	Wi-Fi enabled or not	enabled	disabled
PAUSERESUME	b23	pause/resume feature used or not	used	unused

4.3.4.2 Serial mode definitions

Bit	Value	Description
0	RS232	full duplex standard RS-232
1	RS485F	full duplex RS-485
2	UART	non-inverted 3V (idle high)
3	UART_IDLELOW	inverted 3V (idle low)

4.3.5 Deployment

User settings that do impact the values, storage, or presentation of deployment data for all schedules, or which are otherwise closely related to the deployment.

Item	Size (bytes)	Comment
Section ID	4	0x04000000, "4.0.0.0"
Section size	2	Fixed, 116 as it stands. Includes CRC.
data_format	4	An enumerated code. See **Memory format codes**.
output_format	4	Bit flags enable/disable features of the format. See **Outputformat bit flags**.
status	2	At time of **enable**; one of pending (1), gated (4) or logging (2). [1]
enable_time	8	Logger time when **enable** successful, 64b Unix epoch milliseconds.
start_time	8	64b Unix epoch milliseconds
end_time	8	64b Unix epoch milliseconds
utc_offset	4	milliseconds (signed 32b integer). If the **clock** command reports **offsetfromutc = unknown**, the value is -2147483648 (0x80000000).
simulation_period	4	milliseconds
reserved	1	
reserved	1	
reserved	4	
reserved	4	
wifi_reference_pressure	4	float, nominal pressure at water surface (not fixed, can vary during deployment).
batt_type_internal	2	Enumerated ID code See **Internal battery codes**.

Item	Size (bytes)	Comment
batt_type_external	2	Enumerated ID code See **External battery codes**.
batt_cap_internal	4	float, Joules
batt_cap_external	4	float, Joules
energyused_internal	4	float, Joules at time deployment enabled
energyused_external	4	float, Joules at time deployment enabled
speccond_tempco	4	float
default_temp	4	float
default_pres	4	float
default_atms	4	float
default_dens	4	float
default_salinity	4	float
default_avgsoundspeed	4	float
altitude	4	float
CRC	2	A standard 16b CRC of all Section 4.

[1]At the time of writing the intent is to retain these enumerated status values, and not to condense them to the values now reported by the **deployment status** command. Under that scheme, 'pending' is a special case of 'gated' (gated by time), and 'logging' is now known as 'sampling'.

4.3.5.1 Memory format codes

Code	Name	Description
0	QUERY	(Internal use only: requests current format)

Code	Name	Description
1	FLOAT32	32b IEEE single precision floating point, calibration equation applied.
2	FLOAT64	64b IEEE double precision floating point, calibration equation applied.
3	CALFLOAT64	64b IEEE double precision floating point in the nominal range 0.0 to 1.0, no calibration equation applied.
4	NORMAL	(Internal use only: clears CALFLOAT64 format and restores FLOAT32 or FLOAT64)

4.3.5.2 Outputformat bit flags

Bit	Value	Description
b0	1 / 0	Output does/doesn't include the logger serial number.
b1	1 / 0	Output does/doesn't include a schedule label.
b2	1 / 0	Output does/doesn't include a CRC.
b3	1 / 0	Output encoding is binary (TBD) / ASCII.
b4	1 / 0	Output data type is float64 / float32.
b5	1 / 0	Output does/doesn't include a date and time.
b6 ... b31	n/a	Reserved

Refer to the outputformat command for examples.

4.3.5.3 Internal battery codes

Code	Name	Description
0	NONE	no internal battery at all.
1	OTHER	unspecified battery, not yet supported.
2	LISOCL2	lithium thionyl chloride.
3	LIFES2	lithium iron sulphide.
4	ZNMNO2	generic alkaline.
5	LINIMNCO	lithium-ion rechargeable (UltraFire).

Code	Name	Description
6	NIMH	nickel metal hydride rechargeable.

4.3.5.4 External battery codes

Code	Name	Description
100	NONE	no external power source.
101	OTHER	unspecified external power source.
102	LISOCL2	RBR*fermata* lithium thionyl chloride.
103	ZNMNO2	RBR*fermata* generic alkaline.
104	LIMNO2	RBR*fermette*^{3} CR123A lithium.
105	FERMETTE3_LISOCL2	RBR*fermette*^{3} lithium thionyl chloride.
106	FERMETTE3_LIFES2	RBR*fermette*^{3} lithium iron sulphide.
107	FERMETTE3_ZNMNO2	RBR*fermette*^{3} generic alkaline.
108	FERMETTE3_LINIMNCO	RBR*fermette*^{3} generic alkaline.
109	FERMETTE3_NIMH	RBR*fermette*^{3} generic alkaline.
110	FERMATA_NIMH	RBR*fermata* nickel metal hydride (rechargeable) 12V (12s4p).
111	FERMATA_LISOCL2	RBR*fermata* lithium thionyl chloride 24V (8s6p).
112	FERMATA_ZNMNO2	RBR*fermata* generic alkaline 12V (12s4p).

4.3.6 Configuration

Top level information about the configuration used for the deployment.

Item	Size (bytes)	Comment
Section ID	4	0x05000000, "5.0.0.0"

Item	Size (bytes)	Comment
Section size	2	As it stands, 72. Includes CRC.
dataset_label	32	NUL-terminated string, padded with 0xFF if needed.
configuration_label	32	NUL-terminated string, padded with 0xFF if needed.
CRC	2	A standard 16b CRC of all Section 5.

4.3.7 Schedules

Organized to allow extraction of individual schedules to construct metadata on a per-schedule basis.

4.3.7.1 Schedule map

Item	Size (bytes)	Comment
Section ID	4	0x06010000, "6.1.0.0"
Section size	2	Variable, depends on number of schedules. Includes CRC.
schedule_count	2	16b number of schedules used in this configuration, S
1st schedule index	2	16b internal index value, 1-based
1st schedule label	32	NUL-terminated string, padded with 0xFF if needed.
1st schedule offset	2	Offset in bytes from the very start of Section 6.1.
2nd schedule index	2	16b internal index value
2nd schedule label	32	NUL-terminated string, padded with 0xFF if needed.
2nd schedule offset	2	Offset in bytes from the very start of Section 6.1.
...	...	...
S th schedule index	2	16b internal index value

Item	Size (bytes)	Comment
S th schedule label	32	NUL-terminated string, padded with 0xFF if needed.
S th schedule offset	2	Offset in bytes from the very start of Section 6.1.
CRC	2	A standard 16b CRC of all Section 6.1.

4.3.7.2 Schedule details

Item	Size (bytes)	Comment
Section ID	4	0x06020100, "6.2.1.0"
Section size	2	Variable, depends on type of schedule. Includes CRC.
internal_index	2	Duplicated from the map so the section can stand alone.
schedule_label	32	Duplicated from the map so the section can stand alone.
mode	1	Enumerated code for sampling mode (continuous, burst, etc.). See **Sampling mode codes**.
flags	4	Stream on/off, storage on/off. Also destination of stream. See **Schedule flags**.
user_group_count	1	Number of user-groups in this schedule.
user_groups	16	Internal indices of user-groups in this schedule, fixed-size list, 0xFF-padded.
FE-group_count	1	(Diagnostic) Number of FE-groups in this schedule.
FE-groups	16	(Diagnostic) Internal indices of FE-bus groups, fixed-size list, 0xFF-padded.
parameters	variable	mode-dependent sampling parameters See Sampling mode parameters • continuous • burst / wave / average / tide • ddsampling • regimes

Item	Size (bytes)	Comment
CRC	2	A standard 16b CRC of all Section 6.2.1.

4.3.7.3 Sampling mode codes

Code	Sampling mode
0	CONTINUOUS
1	AVERAGE
2	TIDE
3	BURST
4	WAVE
5	DDSAMPLING
6	REGIMES
7	unknown (error)

Schedule flags

Bit	Value	Description
b0	1 / 0	Data is / isn't stored to memory.
b1	1 / 0	Data is / isn't streamed to USB.
b2	1 / 0	Data is / isn't streamed to Serial.
b3	1 / 0	Data is / isn't streamed to Wi-Fi.
b4 ... b31	n/a	Reserved

4.3.7.4 Sampling mode parameters

4.3.7.4.1 continuous

Parameter	Size (bytes)	Description
period	4	milliseconds
cast_detection	1	off (0) | on (1)

4.3.7.4.2 Burst sampling (averaging, tides, waves)

Parameter	Size (bytes)	Description
measurementperiod	4	milliseconds
period	4	milliseconds
measurementcount	4	samples

4.3.7.4.3 Directional sampling (ddsampling)

Parameter	Size (bytes)	Description
direction	1	descending (0) | ascending (1)
cast_detection	1	off (0) | on (1)
fast_period	4	milliseconds
slow_period	4	milliseconds
fast_threshold	4	dbar, float
slow_threshold	4	dbar, float

4.3.7.4.4 Regime sampling

Parameter	Size (bytes)	Description
direction	1	descending (0) | ascending (1)
count	1	1 | 2 | 3
reference	1	This is an index of the channel used as a reference pressure for controlling regimes operation. The channel indices are assigned in the same order as the channel labels are reported by the command **channels channellist**. The index of the first channel is 1.
finalboundary	2	units of dbar
boundary1	2	units of dbar
binsize1	2	units of 0.1 dbar
period1	4	milliseconds
boundary2	2	units of dbar
binsize2	2	units of 0.1 dbar
period2	4	milliseconds
boundary3	2	units of dbar
binsize3	2	units of 0.1 dbar
period3	4	milliseconds

4.3.8 User groups

Organized to allow extraction of individual groups to construct metadata on a per-schedule basis.

4.3.8.1 User group map

Only those groups used by this configuration appear in the map.

Item	Size (bytes)	Comment
Section ID	4	0x07010000, "7.1.0.0"
Section size	2	Variable, depends on number of groups. Includes CRC.
user_group_count	2	16b number of user-groups used in this configuration, G
1st user-group index	2	16b internal index value, 1-based
1st user-group label	32	NUL-terminated string, padded with 0xFF if needed.
1st user-group offset	2	Offset in bytes from the very start of Section 7.1.
2nd user-group index	2	16b internal index value
2nd user-group label	32	NUL-terminated string, padded with 0xFF if needed.
2nd user-group offset	2	Offset in bytes from the very start of Section 7.1.
...	**...**	**...**
G th user-group index	2	16b internal index value
G th user-group label	32	NUL-terminated string, padded with 0xFF if needed.
G th user-group offset	2	Offset in bytes from the very start of Section 7.1.
CRC	2	A standard 16b CRC of all Section 7.1.

4.3.8.2 User group details

Details for a specific user group

Item	Size (bytes)	Comment
Section ID	4	0x07020100, "7.2.1.0"
Section size	2	As it stands, 116. Includes CRC.
internal_index	2	Duplicated from the map so the section can stand alone.
group_label	32	Duplicated from the map so the section can stand alone.
schedule_list	4	Bit flags referencing schedules to which this group belongs [1]
fe_group_list	4	Bit flags referencing FE-groups associated with this group [1]
channel_count	2	Total number of channels in the group, including hidden
channel_list	64	User group channel details. An ordered list of internal channel (index-flags) pairs, 0xFFFF-padded
CRC	2	A standard 16b CRC of all Section 7.2.1.

[1] Bit flags for any items associated with this group but which are NOT used by this configuration are not set.

4.3.8.2.1 User group channel details

Item	Size (bytes)	Comment
index	1	Internal index of channel, 1..32
flags	1	b0: 1 stored, 0 not stored. b1: 1 streamed, 0 not streamed. b2: 1 always hidden, 0 may be visible b3: 1 hidden for this group, 0 visible b4: 1 derived, 0 measured. b5...b7: reserved (0)

4.3.9 Module group

If present, used for diagnostic purposes.

4.3.9.1 Module group map

Item	Size (bytes)	Comment
Section ID	4	0x08010000, "8.1.0.0"
Section size	2	Variable, depends on number of FE-groups defined. Includes CRC.
fe_group_count	2	16b number of FE-groups defined, F
1st FE-group index	2	16b internal index value, 1-based
1st FE-group offset	2	Offset in bytes from the very start of Section 8.1.
2nd FE-group index	2	16b internal index value
2nd FE-group offset	2	Offset in bytes from the very start of Section 8.1.
...	...	...
F th FE-group index	2	16b internal index value
F th FE-group offset	2	Offset in bytes from the very start of Section 8.1.
CRC	2	A standard 16b CRC of all Section 8.1.

4.3.9.2 Module group details

Item	Size (bytes)	Comment
Section ID	4	0x08020100, "8.2.1.0"
Section size	2	As it stands, 18 without the module_list, 82 with it. Includes CRC.
index	2	16b internal index value, 1..16
group_list	4	Bit-flags referencing the associated user-group indices.

Item	Size (bytes)	Comment
channel_list	4	Bit-flags referencing the internal channel indices.
module_list	64	Redundant and optional, but may be convenient; up to 32 FE-bus addresses (16-bit) in order of channel list, 0xFF-padded.
CRC	2	A standard 16b CRC of all Section 8.2.1.

4.3.10 Channels

Internal map, then each channel has its own self-contained subsection, so that individual channels can be extracted to construct metadata on a per-schedule basis.

4.3.10.1 Channel map

Item	Size (bytes)	Comment
Section ID	4	0x09010000, "9.1.0.0"
Section size	2	Variable. Includes CRC.
channel_count	2	Number of channels included in the configuration, C.
1st channel index	2	Internal index value, 1...32
1st channel label	32	NUL-terminated string, padded with 0xFF if needed.
1st channel offset	2	Offset in bytes from the very start of Section 9.1.
2nd channel index	2	Internal index value, 1...32
2nd channel label	32	NUL-terminated string, padded with 0xFF if needed.
2nd channel offset	2	Offset in bytes from the very start of Section 9.1.
...	**...**	**...**
C th channel index	2	Internal index value, 1...32

Item	Size (bytes)	Comment
C th channel label	32	NUL-terminated string, padded with 0xFF if needed.
C th channel offset	2	Offset in bytes from the very start of Section 9.1.
CRC	2	A standard 16b CRC of all Section 9.1.

4.3.10.2 Channel details

Item	Size (bytes)	Comment
Section ID	4	0x09020100, "9.2.1.0"
Section size	2	Variable, depends on channel details. Includes CRC.
internal_index	2	Duplicated from the map so the section can stand alone.
module_address	2	FE-bus address corresponding to this channel.
type_key	16	NUL-terminated string, padded out to 16 bytes with 0xFF; for Ruskin to use as it sees fit, unused by the logger.
channel_label	32	Duplicated from the map so the section can stand alone.
fw_info_size	2	Length of FE-module firmware information string, includes terminating NUL (may be NUL-padded to a multiple of four bytes).
fw_info_string	variable	NUL-terminated string reporting all FE-module firmware information (may be NUL-padded to a multiple of four bytes). Example for L3/L3.5 FE-module: `T = FE-serial-phytoflash-Yield, F = 10.041, r = 4662, H = 1, A = 128`
user_groups	4	Bit flags referencing the internal indices of user-groups to which this channel belongs. [1]
fe_group_list	4	Bit flags referencing FE-groups associated with this channel. [1]
flags	4	Bit mask of channel properties. See Channel property bit flags.
settling_time	4	milliseconds

Item	Size (bytes)	Comment
read_time	4	milliseconds (intended to apply only to power-cycled, 'normal' sampling.)
guard_time	4	milliseconds - time for which the power must remain off once it has been removed.
specifics_count	2	Number of channel-specific information items following calibration coefficients.
specifics_offset	2	Offset in bytes from beginning of section to first channel-specific item.
The following are fixed-size items relating to calibration.		
equation	32	Fixed length character array giving the name of the calibration equation used for this channel; up to 31 printable characters terminated with a NUL (0) character. See Channel equation types.
calibration_date	8	Format: Unix epoch milliseconds
user_offset	4	float32, default 0.0.
user_slope	4	float32, default 1.0.
factory_units	16	NUL-terminated string for display of units, 0xFF-padded - units used by RBR for Factory calibration, not user-modifiable.
user_units	16	NUL-terminated string for display of units, 0xFF-padded - potentially user-specified, for future use; same as factory_units unless modified.
coefficient_count	4	Enhanced to assist with navigation. See Coefficient count breakdown.
Actual calibration coefficients: total size of coefficient storage is not fixed.		
coefficient_c0	4	The **C** and **X** coefficients are single precision IEEE floating point numbers. The **N** cross-reference indicators are signed 32-bit integers.
coefficient_c1	4	
...	**...**	
coefficient_nN	4	
Total size of storage for channel-specific information is not fixed: if 'specifics_count' (above) is zero, there will be nothing here.		

Item	Size (bytes)	Comment
Channel-specific item #1 (if any)	variable	There are various types of information required only for a particular type of channel and which do not apply to other types; these are referred to as "channel;-specific" items. Storage requirements for these items vary widely because the information is so diverse. See Channel specific items.
Channel-specific item #2	variable	Details of the 2nd item which is specific to this channel, as above for 1st item.
...	...	...
Channel-specific item #N	variable	Details of the Nth item which is specific to this channel, as above for 1st item.
CRC	2	A standard 16b CRC of all Section 9.2.1.

[1] Bit flags for any items associated with this channel but which are NOT used by this configuration are not set.

4.3.10.2.1 Channel property bit flags

Bit	Description
b00	unused
b01	Uses MAX11210 A/D converter.
b02	Supports gain switching.
b03	Channel is derived.
b04	High resolution data
b05	Can control Wi-Fi, cast detection, ddsampling.
b06	Can control regimes.
b07	Can be conductivity reference for cast detection.
b08	Must handle a long read-time (will likely become redundant).
b09	Temperature used for conductivity correction.
b10	Is a temperature channel.

Bit	Description
b11	Is an FE-freq channel.
b12	Requires 12V supply for sensor.
b13	Requires extended power-down sequence, uses READY/*BUSY mechanism.
b14	Takes special actions at start and/or end of deployment.
b15	Validity depends on hardware revision.

4.3.10.2.2 Channel equation types

Name	Description
corr_cond3	corr_cond3 - Conductivity corrections for RBRLegato and 6000dbar C and CT cell
corr_irr	corr_irr - Irradiance
corr_irr2	corr_irr2 - generic irradiance and PAR
corr_metsmeth	corr_metsmeth - Temperature correction of METS methane output
corr_metstemp	corr_metstemp - Temperature measured by a METS (methane sensor)
corr_o2conc_garcia	corr_o2conc_garcia - O2 concentration compensated for salinity and pressure
corr_ph	corr_pH - Simple temperature correction of pH
corr_pres2	corr_pres2 - Temperature correction of Pressure
corr_rinkob2	corr_rinkoB2 - Correction of Rinko Dissolved Oxygen using Rinko Temperature sensor
corr_rinkotemp	corr_rinkotemp - Temperature measured by a Rinko DO sensor
cub	cub, or Cubic
deri_bprpres	deri_bprpres and deri_bprtemp, BPR channels
deri_bprtemp	deri_bprpres and deri_bprtemp, BPR channels
deri_depth	depth - derivation of depth from pressure
deri_dyncorrS	deri_dyncorrT and deri_dyncorrS dynamic correction channels

Name	Description
deri_dyncorrT	deri_dyncorrT and deri_dyncorrS dynamic correction channels
deri_o2sat_garcia	deri_o2sat_garcia, Derived O2 saturation from concentration
deri_salinity	pss78 - derivation of Practical Salinity (1978)
deri_seapres	seapres - derivation of sea pressure from pressure
deri_sos	deri_sos, speed of sound
deri_speccond	deri_speccond - derivation of specific conductivity
lin	lin, or Linear
lind	Same as lin, or Linear but using double precision arithmetic, for high resolution channels (e.g. FE-freq).
none	
optic2	optic2 - optical parameters measured by a Satlantic OCR sensor
qad	qad, or Quadratic

4.3.10.2.3 Coefficient count breakdown

Bit	Description
b00...b07	Total number of coefficients.
b08...b15	Number of **C** coefficients.
b16...b23	Number of **X** coefficients.
b24...b31	Number of **N** cross-references.

4.3.10.2.4 Channel specific items

Item	Size (bytes)	Description
item_type	1	An enumerated code for the type of information.
item_size	2	Total size of item in bytes.
item_spare	2	Unused
item_data	variable	Currently three types of information are supported: • Gain switching data (item_type = 3) • Frequency settings (item_type = 128) • Sensor key-value pairs (item_type = 2)

Gain switching data

item_type = 3

Item	Size (bytes)	Description
mode	1	Gain switching mode, none=0, manual=1, auto=2.
gain_count	1	Number of gain settings available.
current_gain_value	4	float, gain value in use, minimum gain if auto-ranging.
all_gain_values	4 * gain_count	floats, values of all gains

Frequency settings

item_type = 128

Item	Size (bytes)	Description
integration_time	4	milliseconds
meas_average	1	For over-sampling; sample count
settling_time	2	milliseconds
interp_cos[]	8	Array of 4 signed 16b integers, internal coefficients

Item	Size (bytes)	Description
pot_setting	1	Setting of digital frequency-trimming
xtal_settling	2	milliseconds
cal_date	8	calibration date in Unix epoch milliseconds

Sensor key-value pairs

item_type = 2

Item	Size (bytes)	Description
key_name	variable	NUL-terminated string, length arbitrary, but 0xFF-padded to a multiple of 4 bytes if needed.
value_string	variable	NUL-terminated string, length arbitrary, but 0xFF-padded to a multiple of 4 bytes if needed.

4.4 Event data storage format

4.4.1 Layout

Each event stored in memory occupies 24 bytes and has the following layout.

Timestamp	Schedules	Size	Type	Auxiliary
64 bits (8 bytes)	32 bits (4 bytes)	16 bits (2 bytes)	16 bits (2 bytes)	64 bits (8 bytes)

All individual data items are stored in memory in little-endian format (least significant byte at lower address). Descriptions follow below.

Timestamp: a 64-bit count of milliseconds elapsed since 1970/01/01 00:00:00, commonly referred to as the Unix epoch. It accounts for leap years, but *not* leap seconds, time zones or any other adjustments. Note that although the Unix epoch is used as a reference point, the time does not conform exactly to the definition of "Unix time", which is measured in seconds.

Schedules: a 32-bit mask indicating which schedule(s) the event applies to by the schedule's internal reference, an integer in the range 1 to 32. Bit-0 =1 for Schedule 1, Bit-1 = 1 for Schedule 2, and so on.

Size: a 16-bit value that gives the size of the complete event in bytes. Currently the value is always 24, but different event sizes may be supported in the future.

Type: a 16-bit lookup index indicating what kind of event this is.

Auxiliary: an 8-byte array of supplementary data, some or all of which may be unused. The content depends on the event **Type**. Currently, there are always 8 bytes; any unused bytes are at the end of the array and 'padded' with a value of 0xFF (255).

4.4.2 Event types and auxiliary data

Type	Hex	Description	Auxiliary data
0	0x00	Unknown or unrecognized events	
1	0x01	reserved	
2	0x02	**disable** command received	
3	0x03	Run-time error encountered	[2-byte filename hash code], [2-byte line number], [0xFF], [0xFF], [0xFF], [0xFF]
4	0x04	CPU reset detected	
5	0x05	One or more parameters recovered after reset	
6	0x06	Restart failed: real-time clock (RTC)/ calendar contents not valid	
7	0x07	Restart failed: logger status not valid	
8	0x08	Restart failed: primary schedule parameters could not be recovered	
9	0x09	Unable to load alarm time for next sample	
10	0x0A	Sampling restarted after resetting RTC	
11	0x0B	Parameters recovered; sampling restarted after resetting RTC	
12	0x0C	Sampling finished: deployment end time reached	
13	0x0D	reserved	
14	0x0E	reserved	
15	0x0F	Power source switched to USB	
16	0x10	reserved	
17	0x11	reserved	

Type	Hex	Description	Auxiliary data
18	0x12	reserved	
19	0x13	reserved	
20	0x14	reserved	
21	0x15	reserved	
22	0x16	Power source switched to internal battery	
23	0x17	Power source switched to external battery	
24	0x18	reserved	
25	0x19	reserved	
26	0x1A	reserved	
27	0x1B	reserved	
28	0x1C	Regimes enabled, but not yet in a regime	
29	0x1D	Entered regime 1	
30	0x1E	Entered regime 2	
31	0x1F	Entered regime 3	
32	0x20	End of regime bin	
33	0x21	reserved	
34	0x22	reserved	
35	0x23	reserved	
36	0x24	Battery failed, schedule finished	
37	0x25	reserved	
38	0x26	reserved	

Type	Hex	Description	Auxiliary data
39	0x27	reserved	
40	0x28	reserved	
41	0x29	reserved	
42	0x2A	reserved	
43	0x2B	reserved	
44	0x2C	reserved	
45	0x2D	Regimes; passed final boundary	

5 Channel labels

Channel labels are short strings describing the physical parameter measured. These names are used by the channel command.

A label consists of a dedicated prefix and then a 2 digit unique identifier to ensure that all labels in an instrument are unique.

Below are a list of prefixes used by different channel types.

Parameter	Channel type	Label prefix

Backscatter (RBR*tridente*)	`turb14`
	`turb22`
	`turb24`
		`backscatter_`

Chlorophyll (RBR*tridente)*	`fluo43`	`chlorophyll_`

Conductivity	`cond13`
	`cond17`
	`cond19`
	`cond20`
	`cond21`
	`cond22`
	`cond23`
	`cond24`
	`cond25`
	`cond26`
		`conductivity_`

Count (Regimes bin)	`cnt_00`	`count_`
Depth	`dpth01`	`depth_`

Dissolved Oxygen concentration (RBR*coda* T.ODO)	`doxy23`
	`doxy27`
	`doxy28`
	`doxy33`
	`doxy36`
		`oxygenconcentration_`

Parameter	Channel type	Label prefix

Dissolved Oxygen phase (RBR*coda* T.ODO)	`opt_07`
	`opt_14`
	`opt_15`
	`opt_24`
	`opt_35`
		`odophase_`

Dissolved Oxygen saturation (RBR*coda* T.ODO)	`doxy22`	`oxygensaturation_`

Dissolved Oxygen temperature (RBR*coda* T.ODO)	`temp16`
	`temp17`
	`temp24`
	`temp37`
	`temp51`
		`odotemperature_`

Irradiance (BRR*coda* rad)	`irr_06`	`irradiance_`
Irradiance (RBR*quadrante*)	`irr_08`	`irradiance_`
PAR (BRR*coda* PAR)	`par_06`	`par_`
PAR (RBR*quadrante*)	`par_08`	`par_`
Pressure	`pres24`	`pressure_`
Salinity	`sal_00`	`salinity_`
Salinity (dynamically corrected)	`sal_01`	`salinitydyncorr_`
Sea pressure	`pres08`	`seapressure_`

Parameter	Channel type	Label prefix

Temperature	`temp14`
	`temp19`
	`temp26`
	`temp27`
	`temp32`
	`temp33`
	`temp36`
		`temperature_`

Temperature (Conductivity cel)	`temp22` `temp34`	`conductivitycelltemperature_`
Temperature (Dynamically corrected)	`temp38`	`temperaturedyncorr_`
Turbidity (RBR*tridente)*	`turb12` `turb13`	`turbidity_`
Turbidity (RBR*coda* Tu)	`turb19`	`turbidity_`
Turbidity OBS (RBR*coda* Tu)	`turb20`	`turbidity_obs_`

6 Calibration equations and cross-channel dependencies

In the following section, the various equations available in the instrument are described. Some equations applied to a channel are straightforward (like the linear equation or the temperature equation). Others might refer to other channel readings (cross-channel dependencies) to correct various physical effects (for example, conductivity with pressure and temperature correction). Others are for purely derived channels (for example, the derivation of practical salinity).
For each equation, the following will describe the coefficients used (**c** and **x** group of parameters of the calibration command) and the cross-channel dependencies required (**n** group of parameters of the calibration command).

The primary input to most equations is the raw reading of the channel (sometimes referred to as 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". It might also be an internally normalized reading of a digital sensor (external or internal).

6.1 lin - linear equation

$$output = c_0 + c_1 \cdot R$$

6.2 cub - cubic equation

$$output = c_0 + c_1 \cdot R + c_2 \cdot R^2 + c_3 \cdot R^3$$

6.3 qad - quadratic equation

$$output = c_0 + c_1 \cdot R + c_2 \cdot R^2$$

6.4 tmp - 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

$$X = ln(\frac{1}{R} - 1)$$

Examples

```
>> calibration temperature_00 equation
<< channel temperature_00 equation=tmp
```

Confirm the equation type.

```
>> calibration temperature_00
<< calibration temperature_00 datetime=20251001000000 equation=tmp offset=0.0000000e+000
 slope=1.0000000e+000 c0=3.4652630e-003 c1=-251.34581e-006 c2=2.4693230e-006
 c3=-78.103464e-009
```

Request confirmation of all calibration coefficients.

6.5 corr_pres2 - pressure with temperature correction

For an analogically interfaced pressure transducer, 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 R is the normalized voltage ratio read by the ADC 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 label of the temperature channel near the transducer "T", in the examples pressuretemperature_00 value(**n0**) is the final output value of the temperature channel in °C.
- Pcorr is the corrected output in dbar.

Examples

```
>> calibration pressure_00 datetime=20251001000000 c0=0.2346 c1=120.9873 c2=2. 7356
c3=0.7
```

Set the core coefficients.

```
>> calibration pressure_00 datetime=20251001000000 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 pressure_00
<< calibration pressure_00 datetime=20251001000000 equation=corr_pres2
offset=0.0000000e+000 slope=1.0000000e+000 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=pressuretemeprature_00
```

Request confirmation of all calibration coefficients.

6.6 corr_cond3 - conductivity with pressure and temperature correction

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:

$$Corr = \frac{Craw - x_0 \cdot (value(n_0) - x_5)}{1 + x_1 \cdot (value(n_0) - x_5) + (x_2 \cdot (value(n_1) - x_6) + x_3 \cdot (value(n_1) - x_6)^2 + x_4 \cdot (value(n_1) - x_6)^3)}$$

where

- 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 label of the internal temperature of the conductivity cell channel in °C, in the examples conductivitycelltemperature_00.
  value(**n0**) is the final output value of the internal temperature of the conductivity cell channel in °C.
- **n1** is the label of the pressure channel, in the examples pressure_00.
  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 "**parameters pressure**" command).

Examples

```
>> calibration conductivity_00 datetime=20251001000000 c0=0.2346 c1=153.4873 c2=1.0001
```

Set the core coefficients.

```
>> calibration conductivity_00 datetime=20251001000000 x0=0.2003 x1=0.2943 x2=0.005
x3=0.085 x4=0.0001 x5=15.0280 x6=10.0025
```

Set the cross-channel correction coefficients.

```
>> calibration conductivity_00
<< calibration conductivity_00 datetime=20251001000000 equation=corr_cond3
offset=0.0000000e+000 slope=1.0000000e+000 c0=16.552176e-003 c1=157.61278e+000
c2=1.0000000e+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 x6=10.000000e+000
n0=conductivitycelltemperature_00 n1=pressure_00
```

Request confirmation of all calibration coefficients.

6.7 deri_seapres - derivation of sea pressure from absolute pressure

The sea pressure (also referred to as hydrostatic pressure) is simply the difference between pressure measured underwater and atmospheric pressure.

$$P_{sea} = P - P_{atm}$$

In the form used by the logger, using an RBR*concerto*^{3} C.T.D as an example, this becomes:

$$P_{sea} = value(n_0) - value(n_1)$$

- **n0** is the label of the pressure channel, pressure_00 in the example.
- **n1** is the label of the atmospheric pressure channel; not present in the example (default value, refer to **parameters atmosphere**).

Examples

```
>> calibration seapressure_00 equation
<< channel seapressure_00 equation=deri_seapres
```

Confirm the equation type.

```
>> calibration seapressure_00
<< calibration seapressure_00 datetime=20251001000000 equation=deri_seapres
offset=0.0000000e+000 slope=1.0000000e+000 n0=pressure_00 n1=value
```

Request confirmation of all calibration coefficients.

6.8 deri_depth - derivation of depth from absolute pressure

This derived channel implements a simplified equation for water depth in meters, in which no account is taken of either geographical variations in the Earth's gravitational field, or the variation of water density with depth: both these quantities are treated as constants.

$$Dm = \frac{P - Patm}{p \cdot g}$$

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 label of the pressure channel, pressure_00 in the example.
- **n1** is the label of the atmospheric pressure channel; not present in the example (default value, refer to **parameters atmosphere**).
- **p** is the value set for the density of water using the "**parameters density**" command, g is a fixed constant 0.980665, representing the standard value of acceleration due to gravity, in units which correctly account for pressures being measured in decibars.
- **Dm** is the calculated depth in meters.

Examples

```
>> calibration depth_00 equation
<< channel depth_00 equation=deri_depth
```

Confirm the equation type.

```
>> calibration depth_00
<< calibration depth_00 datetime=20251001000000 equation=deri_depth
offset=0.0000000e+000 slope=1.0000000e+000 n0=pressure_00 n1=value
```

Request confirmation of all calibration coefficients.

6.9 pss78 - derivation of practical salinity

The equation relating salinity to conductivity, temperature and pressure is the Practical Salinity Scale of 1978, often referred to as PSS78 (see Practical salinity of seawater). If the salinity calculation leads to an aberrant value, which generally happens 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 our example:

- **n0** is the label of the temperature channel, temperature_00 in this example.
- **n1** is the label of the absolute pressure channel, pressure_00 in this example.
- **n2** is the label of the conductivity channel, conductivity_00 in this example.
- **n3** is the label 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 reports a small negative value. This does not apply if one of the parameters is already flagged as an error.

Examples

```
>> channel salinity_00 equation
<< channel salinity_00 equation=deri_salinity
```

Confirm the equation type.

```
>> calibration salinity_00
<< calibration salinity_00 datetime=20251001000000 equation=deri_salinity
offset=0.0000000e+000 slope=1.0000000e+000 n0=temperature_00 n1=pressure_00
n2=conductivity_00 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 salinity_00
<< calibration salinity_00 datetime=20251001000000 equation=deri_salinity
offset=0.0000000e+000 slope=1.0000000e+000 n0=temperature_00 n1=value n2=conductivity_00
n3=value
```

6.9.1 Practical salinity of seawater

Since it is not possible to directly measure the absolute salinity of seawater (the ratio of the mass of dissolved material to the mass of seawater), it is necessary to work in terms of practical salinity, which can be determined from measurable properties of seawater.

This is defined in "Algorithms for computation of fundamental properties of seawater", by N. P. Fotonoff and R. C. Millard Jr.:

> *The practical salinity, symbol S, of a sample of sea water, is defined in terms of the ratio K of the electrical conductivity of a sea water sample of 15°C and the pressure of one standard atmosphere, to that of a potassium chloride (KCl) solution, in which the mass fraction of KCl is 0.0324356, at the same temperature and pressure. The K value exactly equal to one corresponds, by definition, to a practical salinity equal to 35.*

The practical salinity of seawater can be calculated from three measurable parameters: electrical conductivity, temperature, and pressure. Each of the three parameters is necessary for the salinity calculation since the electrical conductivity of seawater changes with temperature and pressure. Electrical conductivity of seawater is dependent upon the number of dissolved ions per volume (salinity), as well as the mobility of those ions (affected by temperature and pressure). The accuracy of the salinity 'measurement' depends on the accuracy to which the three principal parameters can be measured.

The Practical Salinity Scale of 1978, endorsed by UNESCO/IAPSO, is currently the world standard for salinity calculation. It is used by all RBR CTD instruments and software for the calculation of seawater salinity, using the equations given below; these are taken from "IEEE Journal of Oceanic Engineering", Vol. OE-5, No. 1, January 1980, page 14. Practical salinity, S, is given by:

$$S = a_0 + a_1 \cdot RT^{0.5} + a_2 \cdot RT^{1.0} + a_3 \cdot RT^{1.5} + a_4 \cdot RT^{2.0} + a_5 \cdot RT^{2.5} + \Delta S$$

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

	a	b	c	d	e
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			

6.10 deri_dyncorrT and deri_dyncorrS - derivation of practical salinity with dynamic correction

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 that implement the dynamic correction equations. 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.

6.10.1 deri_dyncorrT equation

If the sample is acquired or polled 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 label of the supporting marine temperature channel (n0=temperature_00 in the examples below). value(**n0**) is the final output value of this channel in degree Celsius, i.e $\mathbf{T_{meas}}$.

6.10.2 deri_dyncorrS equation

$$\begin{aligned} \text{Salinity(PSU)} &= pss78(C, T_{cell}, P) \\ T_{cell} &= T_{cor} + T_{long} - T_{short} \end{aligned}$$

with

$$T_{long} = \text{ctcoeff}(n) \cdot (T_{cond} - T_{cor})$$

and

$$\begin{aligned} T_{short}(n) &= -b(n) \cdot T_{short}(n-1) + a(n) \cdot (T_{cor}(n) - T_{cor}(n-1)) \\ a(n) &= \frac{4 f_N \cdot \alpha(n) \cdot \tau(n)}{1 + 4 f_N \cdot \tau(n)} \\ b(n) &= 1 - 2a(n) \cdot \alpha(n)^{-1} \\ f_N &= \frac{F_s}{2} \end{aligned}$$

where

$$\begin{aligned} \text{ctcoeff}(n) &= \text{ctcoeff}_a * V_p(n)^{\text{ctcoeff}_e} \\ \alpha(n) &= \alpha_a * V_p(n)^{\alpha_e} \\ \tau(n) &= \tau_a * V_p(n)^{\tau_e} \end{aligned}$$

and

$$\begin{aligned} V_{p_{est}}(n) &= (1-a) * V_{p_{est}}(n-1) + a * (P(n-1) - P(n)) * F_s \\ a &= 1 - e^{-2*pi*V_{p_{f_c}}/F_s} \\ V_p(n) &= \min\{\max\{V_{p_{est}}(n), V_{p_{min}}\}, V_{p_{max}}\} \end{aligned}$$

If the sample is polled or not acquired while actively sampling at a rate faster or equal to 1Hz,

$$T_{short}(n) = 0$$

If the sample is polled 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 label of the conductivity channel. (n0=conductivity_00 in examples below).
  value(**n0**) is the final output value of this channel in mS/cm, i.e. **C**.
- **n1** is the label of the sea pressure channel. (n1=seapressure_00 in examples below).
  value(**n1**) is the final output value of this channel in dbar i.e. **P**.
- **n2** is the label of the C-T lag corrected temperature channel. (n2=temperature_01 in examples below).
  value(**n2**) is the final output value of this channel in °C, i.e. $\mathbf{T_{cor}}$.
- **n3** is the label of the internal temperature of the conductivity cell channel. (n3=conductivitytemperature_00 in examples below).

value(**n3**) is the final output value of this channel in °C, i.e. $\mathbf{T_{cond}}$.

Examples

```
>> calibration temperature_00 datetime= 20251001000000 x0=0.3500
<< calibration temperature_00 datetime= 20251001000000 x0=0.3500
```

Set the temperature coefficients.

```
>> calibration salinity_00 datetime=20251001000000 x0=0.1300 x1=5.9000 x2=1.0200e-00
<< calibration salinity_00 datetime=20251001000000 x0=0.1300 x1=5.9000 x2=1.0200e-00
```

Set the salinity coefficients provided.

```
>> calibration temperature_01
<< calibration temperature_01 datetime=20251001000000 equation=deri_dyncorrT
offset=0.0000000e+000 slope=1.0000000e+000 x0=0.3500 n0=temperature_00

>> calibration salinity_01
<< calibration salinity_01 datetime=20251001000000 equation=deri_dyncorrS
offset=0.0000000e+000 slope=1.0000000e+000 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=conductivity_00 n1=seapressure_00
n2=temperature_01 n3=conductivitytemperature_00
```

6.11 corr_o2conc_garcia - O2 concentration compensated for salinity and pressure

When an instrument is connected to an RBR*coda*^{3} T.ODO, the latter transfers both the foil temperature and the dissolved oxygen concentration (not compensated for salinity). The instrument then calculates the oxygen concentration compensated for salinity and pressure:

$$O2corr = (C_0 + C_1 \cdot O2unc) \cdot exp(S \cdot (Gb_0 + Gb_1 \cdot Ts + Gb_2 \cdot Ts^2 + Gb_3 \cdot Ts^3) + Gc_0 \cdot S^2) \cdot (1 + P \cdot C_2)$$

where

- O2corr is the corrected O2 concentration, compensated for salinity and pressure.
- O2unc is the uncompensated O2 concentration returned by the RBR*coda*^{3} T.ODO.
- 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

$$Gb_0 = -6.24097e-3$$
$$Gb_1 = -6.93498e-3$$
$$Gb_2 = -6.90358e-3$$
$$Gb_3 = -4.29155e-3$$
$$Gc_0 = -3.11680e-7$$

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 label of the water temperature channel
- **n1** is the label of the salinity channel
- **n2** is the label of the pressure channel
- **n3** is the label of the atmospheric pressure channel (set to **value** to use **parameters atmosphere**)
  Examples

```
>> calibration oxygenconcentration_00
<< calibration oxygenconcentration_00 datetime=20171201000000 offset=0.0000000e+000
slope=1.0000000e+000 c0=0 c1 = 1 c2=3.25e-5 n0=temperature_00 n1=salinity_00
n2=pressure_00 n3=value
```

Request confirmation of all calibration coefficients.

6.12 deri_o2sat_garcia - derivation of O2 saturation from concentration

When an instrument is connected to an RBR*coda*^{3} T.ODO. The RBR*coda*^{3} T.ODO transfers both the foil temperature and the dissolved oxygen concentration (not compensated for salinity). The instrument 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 µMol/L, compensated for salinity.
- Patm is atmospheric pressure.
- 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

$$Ga_0 = 2.00856$$
$$Ga_1 = 3.22400$$
$$Ga_2 = 3.99063$$
$$Ga_3 = 4.80299$$
$$Ga_4 = 9.78188e - 1$$
$$Ga_5 = 1.71069$$
$$Gb_0 = -6.24097e - 3$$
$$Gb_1 = -6.93498e - 3$$
$$Gb_2 = -6.90358e - 3$$
$$Gb_3 = -4.29155e - 3$$
$$Gc_0 = -3.11680e - 7$$

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 label of the concentration channel, already compensated for salinity.
- **n1**, the label of the water temperature channel.
- **n2**, the label of the salinity channel.
- **n3**, the label of the atmospheric pressure channel (set to **value** in order to use **parameters atmosphere**).
  Examples

```
>> calibration oxygensaturation_00
<< calibration oxygensaturation_00 datetime=20171201000000 offset=0.0000000e+000
slope=1.0000000e+000 n0=oxygenconcentration_00 n1=temperature_00 n2=salinity_00 n3=value
```

Request confirmation of all channel indices.

6.13 deri_sos - derivation of speed of sound

Full, in-situ derivation of speed of sound requires salinity, hydrostatic pressure and temperature. 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 is a series of polynomials combined in various ways: 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. 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 label of the temperature channel, temperature_00 in this example
- **n1** is the label of the sea pressure channel, seapressure_00 in this example
- **n2** is the label of the salinity channel, salinity_00 in this example.

Examples

```
>> calibration speedofsound_00
<< calibration speedofsound_00 datetime=20251001000000 equation=deri_sos
offset=0.0000000e+000 slope=1.0000000e+000 n0=temperature_00 n1=seapressure_00
n2=salinity_00
```

Request confirmation of all channel indices.

6.14 deri_speccond - derivation of 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 **corr_cond3**).
- **T** is the temperature used for correction, in °C.
- **K0** is a units correction factor.
- **K1** is a temperature coefficient.

In the calibration settings for the **scon00** derived channel type, the channel cross-reference label 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 parameters command, using the **speccondtempco** parameter. If this parameter is never explicitly set, the default value used is 0.0191.

7 Information, warning, and error codes

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 either **ERR-** (for errors) or **WRN-** (for warnings), followed by a 3-digit decimal number, padded with leading zeroes if necessary. An (currently fictitious) example would be **ERR-065**.

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 contain a variable element, represented here by a '*<place_holder>*'.

The errors have been categorized according to their very broad root cause, which will be one of the following:

- Wrong usage - incorrect specification of the command or one of its parameters, or inappropriate use of a parameter value.
- Change in instrument condition - an previously valid operation can no longer be performed because the state of the instrument has changed in some way.
- Factory misconfiguration - the instrument's internal settings are in an unexpected state.
- Hardware failure - a problem with the instrument has prevented the command from succeeding. One possible course of action to remedy this is to apply a full hardware reset (see Tips for system integrators).

7.1 List of error and warning messages

7.1.1 Warning

Code	Text	Description
WRN-305	storage access already at selected location	The **storage access** command was sent to change to a state the instrument is already in.
WRN-408	instrument was already enabled	The current deployment has not finished; no change will be made to the instrument's state, and all command parameters will be ignored.
WRN-435	instrument state is already disabled	The **disable** command was sent when the instrument is already disabled.

7.1.2 Error

Error code	Reported description	Root cause
ERR-101	command parser busy	Wrong usage
ERR-102	invalid command '*<unknown-command-name>*'	Wrong usage
ERR-104	feature not yet implemented	Wrong usage
ERR-105	command prohibited while logging	Wrong usage
ERR-107	expected argument missing	Wrong usage

Error code	Reported description	Root cause
ERR-108	invalid argument to command: '<*invalid-argument*>'	Wrong usage[1]
ERR-109	feature not available	Wrong usage
ERR-110	buffer full	Wrong usage/Hardware failure
ERR-111	command failed	Hardware failure
ERR-114	feature not supported by hardware	Wrong usage
ERR-115	syntax error '<*invalid-argument*>'	Wrong usage
ERR-116	parse error '<*invalid-argument*>'	Wrong usage
ERR-117	'<*invalid-qualifier*>' is not a known qualifier	Wrong usage
ERR-118	invalid qualifier '<*invalid-qualifier*>'	Wrong usage
ERR-119	missing qualifier	Wrong usage
ERR-120	'<*label*>' is already in use	Wrong usage
ERR-121	no qualified objects	Wrong usage
ERR-122	value '<*value-entry*>' is too long	Wrong usage
ERR-123	'<*value-entry*>' is a keyword and cannot be used as a value	Wrong usage
ERR-124	'<*value-entry*>' cannot be set multiple times	Wrong usage
ERR-125	'<*parameter*>' is read only	Wrong usage
ERR-126	arguments result in ambiguous command	Wrong usage
ERR-127	no value has been set	Wrong usage
ERR-129	no more than <*max_count*> entries allowed in list	Wrong usage
ERR-302	download tracking failed, specify all parameters	Change in instrument condition

Error code	Reported description	Root cause
ERR-405	failed to enable for logging	Hardware failure
ERR-406	cannot 'pause' while not logging	Wrong usage
ERR-407	cannot 'resume' unless paused	Wrong usage
ERR-410	no sampling channels active in group '<*group-label*>'	Wrong usage
ERR-411	period not valid for selected mode	Wrong usage
ERR-413	period too short for serial streaming in schedule '<*schedule_label*>'	Wrong usage
ERR-416	wrong regimes settings	Wrong usage
ERR-419	calibration coefficients are missing	Factory misconfiguration
ERR-420	required channel is turned off	Wrong usage
ERR-430	empty schedule list in configuration '<*configuration-label*>'	Wrong usage
ERR-431	dataset limit of '<*max-count*>' reached, delete dataset(s) to make space	Wrong usage
ERR-432	if simulating, only one schedule allowed	Current L35 restriction, may be lifted in future
ERR-433	no start time specified for gating by time	Wrong usage
ERR-434	**starttime** accessible only if gating by time	Wrong usage
ERR-436	instrument was already enabled with different settings	Wrong usage
ERR-437	invalid argument '<*invalid-argument*>', data storage is not available	Wrong usage
ERR-438	channels on sensor '<*sensor-label*>' split across schedules	Wrong usage
ERR-439	no sampling groups active in schedule '<*schedule_label*>'	Wrong usage

Error code	Reported description	Root cause
ERR-440	incompatible periods for schedules with common channels, '<*schedule_label_a*>' and '<*schedule_label_b*>'	Wrong usage
ERR-441	config '<*config_label*>' has too many channels/sensors for given sampling rate(s)	Wrong usage
ERR-442	config '<*config_label*>' streaming too much data for current baud rate	Wrong usage
ERR-501	item is not configured	Factory misconfiguration
ERR-505	no channels configured	Factory misconfiguration
ERR-601	no calibration for channel '<channel-index>'	Factory misconfiguration

[1]Error ERR-108 is typically caused by the user supplying an incorrect argument to a command. However, it can also be caused by supplying a valid argument in the wrong position for those commands that require some arguments to be in a certain order. For example:

1. **instrument power internal voltage** is correct usage.
2. **instrument power voltage internal** is not.

8 Revision history

Revision No.	Release Date	Notes
A	13-February-2025	Initial Gen4 command reference release
B	9-October-2025	Corrected typo in Quick start. Corrected clock section to indicate when the clock can be changed. Percent added as special character added in Parameter naming constraints. **all**, **erase** and **real** critical parameter added to Parameter naming constraints. id command behaviour corrected. id4 command added. **instrument dump** command typo in description corrected. pcba command examples corrected. download command examples corrected. sensor command updated. pause command added. resume command added. tmp - temperature examples added. deri_seapres - derivation of sea pressure from absolute pressure examples updated. deri_depth - derivation of depth from absolute pressure examples updated. pss78 - derivation of practical salinity examples updated. corr_pres2 - pressure with temperature correction examples updated. deri_dyncorrT and deri_dyncorrS - derivation of practical salinity with dynamic correction examples updated. deri_sos - derivation of speed of sound examples updated. Information, warning, and error codes have been updated. • ERR-441, and ERR-442 added Deprecated the use of the derived parameter in the channel command.

9 Appendix

9.1 Critical parameter keywords which cannot be used as a label

above
absolute
address
aggregate
all
allindices
alllabels
allowed
altitude
ascii
ascending
atmosphere
auto
aux1
aux1_active
aux1_all
aux1_hold
aux1_setup
aux1_sleep
aux1_state
availablebaudrates
availablefastperios
availablegains
availablemodes
availableproperties
availabletypes
average
avgsoundspeed
eeprom
enable
enabled
endaction
encoding
endtime
episodelog
equation
erase
eraseall
erasing
error
eventcount
events
eventsize
eventstart
examine
external
factory
factoryperiodlimit
factoryunits
failed
false
fastperiod
fastthreshold
ebaud
fefreq
fefreq_sensor_settlin
period
period1
period2
period3
permission
permissions
permit
pn
poll
polling
pollpoweroffdelay
power
powerdownbusy
powerexternal
powerfail
powerinternal
poweroffdelay
powerondelay
presdyncorr
pressure
prompt
properties
qad
rapidfill
read
readdata
readtime
real

badresponse
batterytype
baudrate
below
ble
binary
binsize1
binsize2
binsize3
boundary1
boundary2
boundary3
bsl
buspwr
burst
burstcount
burstinterval
bytecount
bytestart
calfloat64
calibration
capacity
castdetection
cellcount
cellformat
channel
channellabel
channellist
channelorder
channels
cleanmemory
fefreq_xtal_settling
fepassthrough
filter
finalboundary
finished
flash
float32
float64
flush
fram
freq
full
fullandstopped
fw
fwlock
fwtype
fwversion
gain
gainswitch
gate
gated
getall
group
grouplist
groups
guardtime
hardwarefeaddress
help
hidden
high
highres
reboot
reference
reg
regimes
regimetrig
remaining
reset
resume
rs232
rs485f
rs485h
rtos
running
Ruskin
salinity
samplecount
samplesize
samplestart
sampling
save
schedule
scheduled
schedulelabel
schedulelist
schedules
scheduletype
seapressure
sensor
sensorchannel
sensorpoweralwayson
segmented

clearcount
clock
closed
coefficient
code
condition
condtemp
conductivity
config
configlist
configs
confirmation
continuous
command
corr_cond3
corr_irr
corr_irr2
corr_metsmeth
corr_metstemp
corr_o2conc_garcia
corr_ph
corr_pres2
corr_rinkoB2
corr_rinkotemp
count
crc
create
cub
data
dataset
datasetlist
hw
hwrev
id
ignore
inactive
index
info
inputtimeout
instrument
internal
interval
invalid
io
L2Flash
label
led
lin
lind
lines
link
list
lock
log
logging
low
max11210
maxcount
mcu
memoryusersize
meta
missing
serial
settings
settlingtime
sim0
sim1
sim2
sim3
simulateddata
simulation
size
sleep
sleepafter
sleeping
slope
slowperiod
slowthreshold
source
smerror
smtimeout
sn
speccondtempco
startaction
startimmediate
starttime
state
status
stopped
storage
storagemode
stream
streamserial

datasets
datatype
datetime
datetimeformat
ddsampling
default
delay
delete
delta
denied
density
deployment
deri_bprpres
deri_bprtemp
deri_depth
deri_dyncorrS
deri_dyncorrT
deri_o2sat_garcia
deri_salinity
deri_seapres
deri_sos
deri_speccond
derived
descending
device
devicesegment
devicesegments
devmode
direction
directional
disable
mode
model
module
modulelist
na
nand
need12v
no
none
noresponse
normal
notblank
notimeout
nvflash
nvram
off
offset
offsetfromutc
on
open
operatingtime
optic2
outputformat
outputmode
p1
p2
p3
p4
parameters
path
pattern
streamusb
success
sustained
temperature
thresholding
tide
time
timeout
timetoepisode
tmp
tristate
true
twistactivation
type
uart
uart_idlelow
uarttest
uniform
unknown
usb
used
userunits
uvled
value
valve
valvesegment
valvesegments
verify
version
violations
visible

disabled

discover

download

dump

duration

e1

patternwrite

pause

paused

pauseresume

pcba

pending

voltage

warning

wave

wifi

wifitrig

write

yes