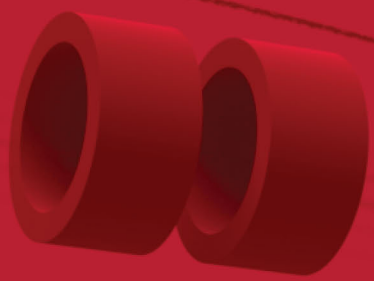
RBR

COMMAND REFERENCE

RBR*coda* OpH



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 instrument via the serial port (RS-232 or RS-485).

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 serial link, this first character may not be completely received by the instrument due to the non-zero wake-up time required, and it may be seen as a garbage character. However, the instrument itself ignores all garbage characters received immediately after wakeup, 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 that 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 instrument will start to wake up.
% [10ms pause]
% The instrument 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 instrument 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 instrument has processed the command and sent the response, any other outputs will be suppressed. Outputs 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 that would have been streamed but has been suppressed due to output blanking is dropped, so it will never be streamed.

The output blanking behaviour does *not* apply to the very first (potential) command received after the instrument 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 instrument 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 minimise 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 instrument has no other tasks to perform the entire instrument will enter a low power sleep mode.

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:

- Timeout protection, 10 seconds after the last parameter modification
- Successfully enabling the instrument to sample
- 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.

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 of the same object; 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 (code 128+)	' ' (space)	'"' (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)	`` ` `` (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. The potential command has been received once the instrument sees a termination character; either one of <**CR**> (0x0D) or <**LF**> (0x0A). Combinations of the two characters are dealt with as follows:

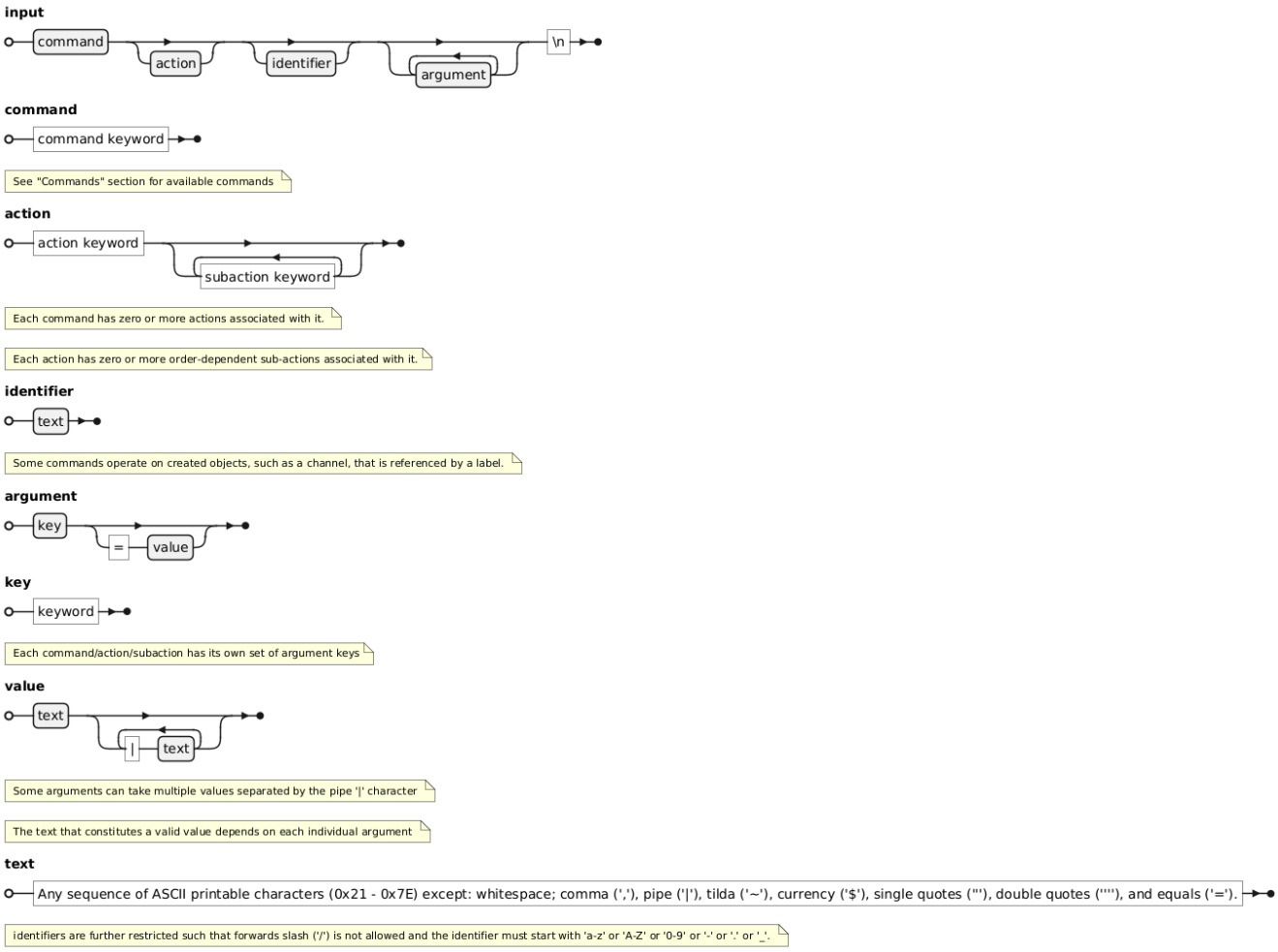
```
>> <CR><LF>
<< ready:

>> <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 instrument prompt, and a total of two prompts are sent (see also the **settings prompt** command).

1.1.4.2 Case sensitivity

In general, the instrument 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 instrument responses, do not assume that the case of the output will match the case of the input: see also Parsing instrument responses.

1.1.4.3 Grammar



1.1.4.4 Common error messages

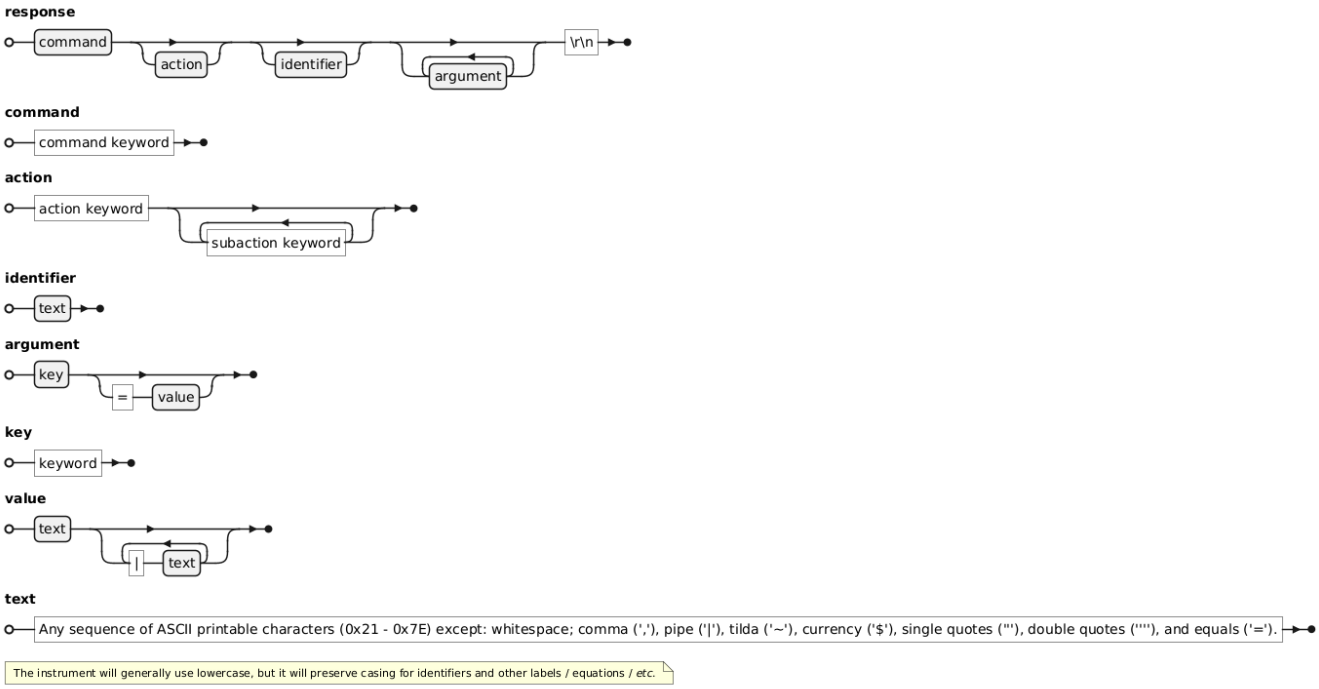
The received message may or may not form a valid command; errors detectable by the instrument will vary from one command to another, but some of the common, general errors include:

- 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 instrument 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 instrument commands:

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

  ```
  >> INSTRUMENT STATE
  << instrument state=disabled
  ```

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

- 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,

  ```
  >> instrument
  << instrument state=disabled sn=999999 model=SL4 pn=9999999revA fwversion=1.0.0
  semver=1.0.0 fwtype=131 fwlock=off datatype=float32 name=SL4
  ```

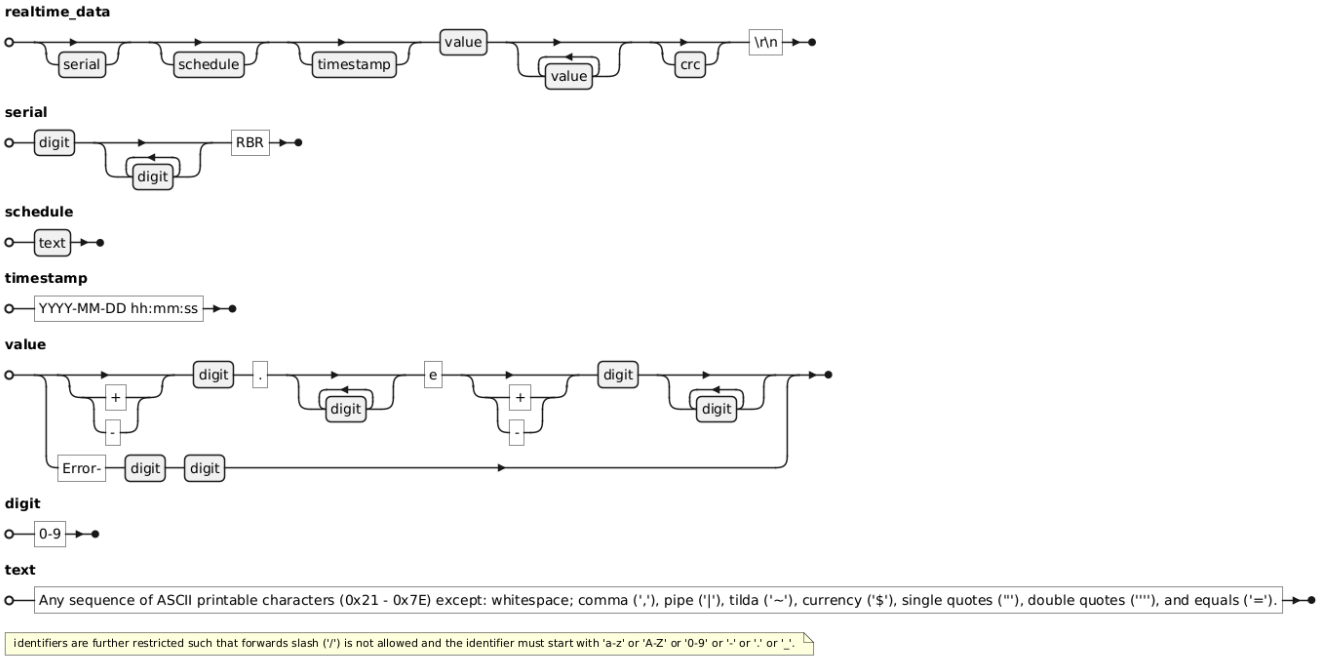
  is the current behaviour, but a future version might respond as follows:

  ```
  >> instrument
  << instrument state=disabled sn=999999 model=SL4 pn=9999999revA fwversion=1.0.0
  semver=1.0.0 fwtype=131 fwlock=off datatype=float32 name=SL4
  futureparameter=132120576
  ```

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

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 instrument is used in a different configuration. For example, setting values via the parameters command might behave as follows:

```
>> parameters pressure
<< parameters pressure=10.1324997

>> parameters pressure=10.1327
<< parameters pressure=10.1327000
```

  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,). Recognising 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, 20260401120000 represents noon on 1st April 2026. 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 that 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, the 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 instrument 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 begin interacting with the instrument.

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

Scheduled samples are configured using various commands. Scheduled samples can be streamed directly out of the instrument in realtime; see the stream parameters of the schedule command. Gen4 instruments have the ability to sample with different measurement parameters according to different schedules, for this we need to understand the concepts of **channels**, **groups**, **schedules**, and **configurations**. These will be described 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 instrument depth in the water.
- One or more schedules are collected together into a configuration.
- The instrument is enabled using one of these configurations, and it will execute the associated schedule(s) to acquire data.

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, Gen4 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, e.g., pressure, the instrument has exactly one channel. Instrument channels may also be available for physical properties that are not measured directly but are derived by calculation from measured channels, e.g., salinity, which is derived from conductivity, temperature, and pressure. In summary:

- A channel represents a single parameter of interest recorded by the instrument. 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 Gen4.
- Users can not create or delete channels. Modification of channel details is limited to its calibration or other specialised parameters (e.g., gain, if applicable).
- Channels are created and assigned a label at the factory when the instrument is built.
- Any channel can be a member of more than one **group**.
- One **configuration** is selected when the instrument 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, e.g., `temperature_00`.

Associated command(s):

channel

calibration

sensor

2.1.3 Groups

- A group is a collection of one or more instrument 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 instrument 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 instrument 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 instrument 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 instrument 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 instrument when it is enabled.
- The instrument can maintain a pool of configurations; when the instrument is enabled, the user specifies exactly one configuration that determines how the instrument will behave during the deployment.
- The user has complete control over the creation, content, and deletion of configurations; if necessary, the instrument 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 instrument.
- The selected configuration forms a subset of the metadata for the deployment, and a copy is stored in memory when the instrument is enabled.

Associated command(s):

config

2.2 Serial streaming from serial port

2.2.1 Setting the correct baudrate

Here we set the baudrate 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.
```

2.2.2 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). The **instrument outputformat** command indicates which channels will be reported and sets the type of output format to be used.

With an RBR*legato*4 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
<< enable config=default_config 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
```

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:

- serial

Examples

```
>> 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

Description

This command can be used to either report or set the parameters which apply to the serial link. 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 instrument is to recognise it.

Modifications to parameters may not be made while logging is enabled; this is to avoid disturbing any realtime 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 instrument'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.
- **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|57600|38400|19200|9600|4800
```

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. Sensor channels used for a poll command will still be shut down.

> If settings **confirmation=on** then 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.**
- **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 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.

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 realtime.

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, **RBRsolo4**
- **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*solo4*, 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 instrument

Usage

```
>> instrument [ state | sn | model | name | 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*> is 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*> is the serial number, it is always reported using six digits, padded with leading zeroes if necessary; for example **092431**.
- **model** is the model name of the instrument; for example, **RBRconcerto4**.
- **pn=**<*part_number*> is the RBR part number of the instrument.
- **fwversion** is the 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 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.
- **name** is the extended model name of the instrument; for example, **RBRsolo^4_T.D!fast32**.
- **fwtype=**<*firmware_type*> is a read-only parameter giving the firmware type code. Possible values are:
  - **140** for SEN4 instruments

Examples

```
>> instrument
<< instrument state=disabled sn=210000 model=RBRsolo4 pn=0123456revA fwversion=1.0.0
fwtype=130 fwlock=off datatype=float32 name=RBRsolo^4_T.D!fast32
```

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
<< enable config=profiling storagemode=normal state=enabled
>> instrument
<< instrument state=enabled sn=210000 model=RBRsolo4 pn=0123456revA fwversion=1.0.0
fwtype=130 fwlock=off datatype=float32 name=RBRsolo^4_T.D!fast32
```

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=RBRsolo4 pn=0123456revA fwversion=1.0.0
fwtype=130 fwlock=off datatype=float32 name=RBRsolo^4_T.D!fast32

>> disable
<< disable state=disabled

>> instrument
<< instrument state=disabled sn=210000 model=RBRsolo4 pn=0123456revA fwversion=1.0.0
fwtype=130 fwlock=off datatype=float32 name=RBRsolo^4_T.D!fast32
```

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=RBRsolo4 pn=0123456revA fwversion=1.0.0
fwtype=130 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.

3.3.2.1 factory

Usage

```
>> instrument factory reset
```

Security

Unsafe

Description

Returns the instrument's sampling configuration to a factory-set state. *ALL* user-defined data relating to configurations, schedules and groups will be lost, although historical datasets stored in the instrument's memory are not affected.

There is only one argument to this command, which is required:

- **reset** executes the factory reset operation.

On success, the command responds with **instrument factory reset**.

The factory-set configurations could vary from one instrument to the next, but will typically be a single schedule running the simplest sampling mode (usually continuous), operating on a single group that contains all channels. All items will have factory-set names, which can be discovered using the **config list**, **schedule list**, and **group list** commands. Once the factory-set configuration has been restored, it can be used as a starting point and modified to suit the user's requirements.

The command is Unsafe; it can not be executed while logging is enabled.

Examples

```
>> instrument factory reset
<< instrument factory reset
```

3.3.2.2 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 realtime 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*> ttt <*value1*> <value2> … <*valueN*><CR><LF>

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

sch_fast_CTD 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 instrument'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 is 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 CRC-CCITT-FALSE 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.
- **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) with 9 significant digits

printed or **float64** (IEEE double precision floating point) with 15 significant digits printed, 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

1. 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.

To remove the schedule label from the format, simplifying the output when only one schedule is used:

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

Format:

ttt <value1> <value2> … <valueN><CR><LF>

Three-channel instrument example:

125 38.6671142e+000 22.0217124e+000 1.95962418e+003<CR><LF>

Add the serial number preamble to the format:

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

Format:

RBR <*serial_number*> <*schedule_label*> ttt <*value1*> <*value2*> … <*valueN*><CR><LF>

Three-channel instrument example:

RBR 142152 sch_fast_CTD 125 38.6671142e+000 22.0217124e+000 1.95962418e+003<CR><LF>

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

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

Format:

<*schedule_label*> ttt <*value1*> <*value2*> … <*valueN*> <*CRC*><CR><LF>

Three-channel instrument example:

sch_fast_CTD 125 38.6671142e+000 22.0217124e+000 1.95962418e+003 0xAEB5<CR><LF>

3.3.2.3 power

Usage

```
>> instrument power [ source ]
```

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

Security

Unsafe.

Description

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

- **source** is a read-only parameter which responds with one of the following names:
  - **external** the instrument is drawing power from an external power source.
- **external** is a sub-command used to access all parameters associated with the instrument'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 source external**

❌ **instrument power external source**

For more details on the sub-command, refer to the external option page.

Examples

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

external

Usage

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

Security

Unsafe

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 instrument 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*fermette3* battery packs are provided by RBR; 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*fermette3*) fermette3_lifes2 **(Li-FeS_2-equipped RBR*fermette3*)** **fermette3_znmno2** (Zn-MnO_2-equipped RBR*fermette3*) fermette3_linimnco **(Li-NiMnCo-equipped RBR*fermette3*)** **fermette3_nimh** (NiMH-equipped RBR*fermette3*) fermata_nimh **(NiMH-equipped RBR*fermata)*** **other** none**
- **used** [**=0**] reports the accumulated energy used from the external power source since the value was last reset. 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 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
```

3.3.2.4 reboot

Usage

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

Security

Open

Description

This command executes a instrument 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 instrument system will be reset in the same way that cycling power to the instrument 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.

Examples

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

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

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

Successfully resets the instrument 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 instrument, the message will not be seen.

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

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

3.3.3 pcba

Usage

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

```
>> pcba <pcba_label> [ sn | pn | fwversion | 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 debugging 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.

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

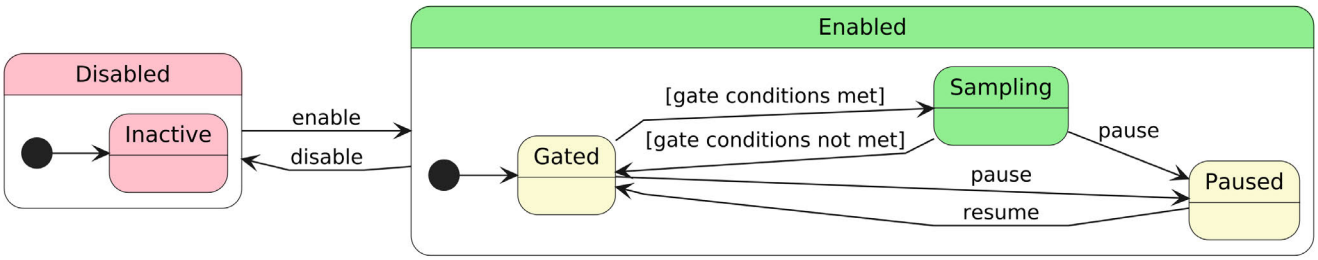
- **sn** reports the serial number of the pcba.
- **pn** reports the **pcba**'s part number.
- **fwversion** reports the **pcba**'s current firmware version.
- **address** reports the base address used to communicate with the **pcba** over the bus.

```
>> pcba 0123456_00
<< pcba 0123456_00 sn=123456 pn=0123456revA fwversion=1.1.1 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 instrument'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 instrument 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 instrument'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 represents 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> [ 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.

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, 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.
- **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
<< verify config=profiling storagemode=normal state=enabled
```

The programmed schedules are valid and the instrument would be enabled.

```
>> instrument state
<< instrument state=enabled
>> verify 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.

```
>> instrument state
<< instrument state=enabled
>> verify 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> [ storagemode=normal | calibration ]
```

Security

Open.

Description

Enables the instrument 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.

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, 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 instrument 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 instrument 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.
- **storagemode=normal** | **calibration** confirming the data storage mode to be used

The command may succeed, but with 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
<< enable config=profiling storagemode=normal state=enabled
```

The programmed schedules are valid and the instrument has been enabled.

```
>> enable config=pH_cal storagemode=calibration
<< enable config=pH_cal 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.

```
>> instrument state
<< instrument state=enabled
>> enable config=pH_cal 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 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** is the state of the instrument, it 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 instrument 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 state
<< instrument state=enabled
>> disable
<< disable state=disabled
```

The instrument was enabled, and has now been disabled.

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

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

3.5 Configuration information and calibration

3.5.1 channel

Usage

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

```
>> channel <channel_label> [ type | address | settlingtime | readtime | guardtime |
userunits | grouplist | sensor | nature ]
```

Security

Unsafe

Description

The channel 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 channel 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 instrument 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 instrument will not turn the channel back on again until this delay has expired.
- **userunits** is a short text string giving the units in which processed data is normally reported from the instrument; 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 **true** or **false** to indicate whether the channel is a derived channel (**true**) or a measured channel (**false**). This is an intrinsic property of the channel **type**, and can not be modified: it is for information only.
- **nature** reports the channel’s nature. It is for information only. There are four distinct categories:
  - **measured** indicates the channel is of scientific value and it is measured directly.
  - **derived** indicates the channel is derived from **measured** channels. i.e. Salinity.
  - **system** indicates the channel pertains to internal affairs of the instrument.
  - **raw** indicates the channel reports a raw uncalibrated value.
- **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.
- **sensor** reports the <sensor_label> for a sensor with which the channel is associated. See the sensor command for more details.

3.5.1.1 Custom attributes

A channel may carry additional **custom attributes** beyond the fixed parameters described above. A custom attribute is a free-form `<key>=<value>` pair, assigned during factory configuration, that the instrument stores but does not otherwise interpret—a place to record channel metadata for which there is no dedicated parameter, such as a sensor's wavelength. Any custom attributes defined for a channel are reported after its standard parameters in the response to **channel** <*channel_label*>, and an individual attribute can be read with **channel** <*channel_label*> <*key*>. End users can read custom attributes, but they are created, modified, and deleted only in Factory Mode.

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
```

3.5.2 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 instrument 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 instrument. **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 instrument 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 instrument 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 instrument which input channels to use.

  There are two special cases when the value of an "**n**" is not a channel label: "**value**" or “**internal**”. value **can only be set at the factory, and applies when an equation requires a correction term using a parameter which the instrument does not measure. In this case the default parameter set by the command parameters will be used. internal** indicates that a calibration is dependent on an internal raw parameter not exposed through the available channel list.

- **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 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 for proper handling.

Please refer to the section Calibration Equations and Cross-channel Dependencies for a complete list of the equations which the instrument 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_01 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.

3.5.3 sensor

Usage

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

```
>> sensor <sensor_label> [sn | pn | fwversion | fwtype | name | 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 summarise 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.
- **fwversion** reports the sensor's current firmware version.
- **fwtype** reports the sensor's current firmware type.
- **channellist** [= <*channel_label_list …*>] reports a list of instrument channels that are associated with this sensor. Labels in the list are separated by a pipe character ('|'), with no spaces.
- **name** is the extended model name of the sensor; for example, **RBRcoda_T.ODO!fast**.

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 fwversion=na fwtype=na
channellist=conductivity_00|temperature_00
```

3.5.4 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 instrument 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. 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 instrument 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
```

3.5.4.1 create

Usage

```
>> group create <group_label>
```

Security

Unsafe

Description

Groups can be created using 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
```

3.5.4.2 delete

Usage

```
>> group delete <group_label> | all
```

Security

Unsafe

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**.

```
>> 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`.

3.5.5 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

Unsafe.

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 instrument 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. 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** | **off**] is used to report or set the communication link over which data for this schedule will be streamed in realtime during the deployment. At most one link may be active for each schedule. Defaults to **off** when the schedule is created.
- **storage** [**off**] determines whether data for this schedule will be stored in memory during the deployment. 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** | **tide** | **wave**] reports or sets the sampling mode to be used by this schedule. Not all possible modes are available in all instruments.
- <*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.
- The order of <*group_labels*> in the <*group_label_list*> determines the order in which channels will be sent in realtime 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 **schedule** 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.
- In all burst sampling modes, the duration in time of a burst must be less than or equal to the interval between bursts.

== 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*>] is the amount of time between consecutive samples; must comply with the generic constraints on sampling periods given above.
- **castdetection** [**= on** | **off**] enables or disables a feature which automatically detects upcasts and downcasts in profiling deployments, and will generate cast detection events in the stored data. It is advisable to ensure this option is **off** if the instrument is not used as a profiler. The default setting is **off**.

== average / tide / burst / wave modes

In all these modes data is acquired in a series of bursts. The bursts all have the same duration, and occur at regular intervals. The duration in time of a burst must be less than the interval between bursts, and may be calculated in milliseconds as **measurementperiod** * **measurementcount**. In **burst** mode, every sample acquired during the burst is stored in memory, and streamed if realtime output is enabled. In **average** mode, samples acquired during the burst are accumulated and averaged at the end; only a single result corresponding to the average is stored in memory or streamed in realtime. Internally, **wave** and **burst** modes are identical, as are **average** and **tide** modes; the alternative names are used as a cue to the host, allowing the retrieved data to be processed and presented differently. The parameters for the bursting modes are as follows:

- **measurementperiod** [= <*milliseconds*>] is the amount of time between consecutive samples, but applicable only within the burst; must comply with the generic constraints on sampling periods given above.
- **measurementcount** [= <*sample_count*>] is the number of samples to attempt to acquire in a burst.
- **period** [= <*milliseconds*>] is the time interval between the first sample of one burst and the first sample of the burst immediately following.

3.5.5.1 create

Usage

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

Security

Unsafe

Description

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

```
>> schedule
<< schedule count=3 maxcount=16 list=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=off
mode=continuous period=1000 castdetection=off

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

3.5.5.2 delete

Usage

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

Security

Unsafe

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**.

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`.

3.5.6 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 instrument can hold in its pool at any given time. The maxcount will differ depending on your instrument type.
- **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.

```
>> 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.

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 .
- 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
```

3.5.6.1 create

Usage

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

Security

Unsafe

Description

A configuration can be created using the **create** action 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
```

3.5.6.2 delete

Usage

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

Security

Unsafe

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.

```
>> config
<< config 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
<< config 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`.

3.5.7 settings

Usage

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

Security

Open.

Description

Reports or sets the values of miscellaneous settings in the instrument 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

3.5.8 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 instrument as described below.

- **altitude** [=<*value*>] is the height above the seabed in metres at which the instrument 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*>] 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 = 0dbar), 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/cm3	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 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 3-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
Temperature	`temp000`	`temperature_`
Optical signal (RBR*coda* OpH)	`opt_001`	`opt_`
pH (RBR*coda* OpH)	`ph__000`	`ph_`
Corrected pH (RBR*coda* OpH)	`ph__001`	`corr_ph_`

5 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 normalised 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 230, and so is often referred to as a "voltage ratio". It might also be an internally normalised reading of a digital sensor (external or internal).

5.1 lin - linear equation

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

5.2 qad - quadratic equation

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

5.3 cub - cubic equation

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

5.4 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

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

Examples

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

Confirm the equation type.

```
>> calibration temperature_00
<< calibration temperature_00 datetime=
```

Request confirmation of all calibration coefficients.

```
>> 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.

5.5 oph - optical pH

Converts the raw integer ratio R output by a PyroScience optical pH sensor into a pH value, applying in-situ temperature corrections. The result is the pH at the reference ionic strength IS = 0.15 (salinity 7.5 g/L); salinity and pressure corrections are applied downstream by **corr_oph**.

$$pH = c5\,(1 + c4\,(T - 20))\,\log_{10}\left(\frac{c1\,(1 + c0\,(T - 20)) - R}{R - c3\,(1 + c2\,(T - 20))}\right) + \; c7 + c6\,(T - 20) + c8$$

where

- **R** = value(**n0**): raw integer ratio from the PyroScience Results register
- **T** = value(**n1**): in-situ temperature [°C]
- **c0** (top_t): temperature sensitivity of c1 [K^{-1}]
- **c1** (top): upper asymptote of R at 20°C — derived from two-point calibration
- **c2** (bottom_t): temperature sensitivity of c3 [K^{-1}]
- **c3** (bottom): lower asymptote of R at 20°C — derived from two-point calibration
- **c4** (slope_t): temperature sensitivity of c5 [K^{-1}]
- **c5** (slope): modified Nernst slope
- **c6** (pKa_t): temperature sensitivity of c7 [pH K^{-1}]
- **c7** (pKa): apparent pKa at 20°C at the IS = 0.15 reference [pH]
- **c8** (offset): pH offset from a 3rd calibration point [pH]; 0 for a 2-point calibration

Examples

```
>> calibration ph_01 equation=oph c0=-803E-6 c1=1793000E-6 c2=-1108E-6 c3=51000E-6 c4=0
c5=1034000E-6 c6=-16280E-6 c7=8090E-3 c8=0 n0=opt_00 n1=temperature_01
```

Set the calibration coefficients and input channels.

```
>> calibration ph_01
<< calibration ph_01 datetime=20251001000000 equation=oph c0=-803E-6 c1=1793000E-6 c2=-
1108E-6 c3=51000E-6 c4=0 c5=1034000E-6 c6=-16280E-6 c7=8090E-3 c8=0 n0=opt_00
n1=temperature_01
```

Request confirmation of all calibration coefficients.

5.6 corr_oph - optical pH with salinity and pressure correction

Applies salinity (ionic strength) and pressure corrections to the output of an **oph** channel.

$$pH_{corrected} = pH - \frac{1}{2}\,c0\left(\frac{\sqrt{S/50}}{1+\sqrt{S/50}} - 0.2791745 - c1\left(\frac{S}{50} - 0.15\right)\right) - c2 \cdot P$$

where

- **pH** = value(**n0**): pH from the upstream **oph** channel
- **S** = value(**n1**): salinity [g/L] (set to value to use **parameters** salinity)
- **P** = value(**n2**): pressure [dbar] (set to value to use **parameters** pressure)
- **c0**: ionic strength correction coefficient 1
- **c1**: ionic strength correction coefficient 2
- **c2**: pressure sensitivity coefficient [pH $dbar^{-1}$]; 2.337×10^{-5} (tentative)

The salinity correction is exactly zero at S = 7.5 g/L (ionic strength 0.15), the reference at which the upstream **oph** channel reports pH; use S = 7.5 g/L to skip the salinity correction. Use P = 0 to skip the pressure correction.

Examples

```
>> calibration corr_ph_00 c0=970E-3 c1=126E-3 c2=2.337E-5 n0=ph_01 n1=value n2=value
```

Set the correction coefficients and input channels. Use `n1=value` and `n2=value` when no salinity or pressure channel is available.

```
>> calibration corr_ph_00
<< calibration corr_ph_00 datetime=20251001000000 equation=corr_oph c0=970E-3 c1=126E-3
c2=2.337E-5 n0=ph_01 n1=value n2=value
```

Request confirmation of all calibration coefficients.

5.7 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.

$$P_{\text{sea}} = P - P_{\text{atm}}$$

In the form used by the instrument, using an RBR*concerto*^{3} C.T.D as an example, this becomes:

$$P_{\text{sea}} = value(n_0) - value(n_1)$$

- **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=
```

Request confirmation of all calibration coefficients.

```
>> 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.

5.8 deri_depth - derivation of depth from absolute pressure

This derived channel implements a simplified equation for water depth in metres, in which no account is taken of either geographical variations in the Earth's gravitational field, or the variation of water density with depth: both these quantities are treated as constants.

$$Dm = \frac{P - Patm}{p \cdot g}$$

In the form used by the instrument, 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 metres.

Examples

```
>> calibration depth_00 equation
<< channel depth_00 equation=deri_depth
```

Confirm the equation type.

```
>> calibration depth_00
<< calibration depth_00 datetime=
```

Request confirmation of all calibration coefficients.

```
>> 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 Information, warning, and error codes

This is a current, but partial, list of error messages which the instrument 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 instrument are unsuitable for any reason.

Note that some messages may contain a variable element, represented here by a '*<place_holder>*'.

The errors have been categorised 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 - a 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).

6.1 List of error and warning messages

6.1.1 Warning

Code	Text	Description
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.

6.1.2 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

Error code	Reported description	Root cause
ERR-107	expected argument missing	Wrong usage
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-301	memory erase not completed	Hardware failure
ERR-405	failed to enable for logging	Hardware failure

Error code	Reported description	Root cause
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-412	burst parameters inconsistent	Wrong usage
ERR-413	period too short for serial streaming in schedule ‘<*schedule_label*>’	Wrong usage
ERR-414	thresholding interval not valid	Wrong usage
ERR-415	more than one gating condition is enabled	Wrong usage
ERR-416	wrong regimes settings	Wrong usage
ERR-417	no gating allowed with regimes mode	Wrong usage
ERR-418	cast detection needs a pressure/depth channel	Wrong usage
ERR-419	calibration coefficients are missing	Factory misconfiguration
ERR-420	required channel is turned off	Wrong usage
ERR-421	ddsampling mode needs pressure channel	Wrong usage
ERR-423	wrong ddsampling settings	Wrong usage
ERR-430	empty schedule list in configuration ‘<*configuration-label*>’	Wrong usage
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-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 baudrate	Wrong usage
ERR-443	channel '<*channel_label*>' is misconfigured	Wrong usage
ERR-444	period too short for channel '<*channel_label*>'	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
ERR-701	device error: <*details*>	Hardware failure
ERR-702	no devices configured	Factory misconfiguration
ERR-703	device schedule inconsistent	Wrong usage
ERR-704	device is not enabled	Wrong usage
ERR-705	multiple operations not supported: '<extra-operation>'	Wrong usage
ERR-706	discovery incomplete	Hardware failure
ERR-707	unsupported instrument	Wrong usage
ERR-708	unsupported firmware version	Wrong usage

[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 incorrect usage.

7 Revision history

Revision No.	Release Date	Notes
A	30-July-2026	Initial release

8 Appendix

8.1 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	availablefastperiods	availablegains	availablemodes
availableproperties	availabletypes	average	avgsoundspeed
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	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	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	disabled	discover	download
dump	duration	e1	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
febaud	fefreq	fefreq_sensor_settling	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	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	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
patternwrite	pause	paused	pauseresume
pcba	pending	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	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
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
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
voltage
warning
wave
wifi
wifitrig
write
yes