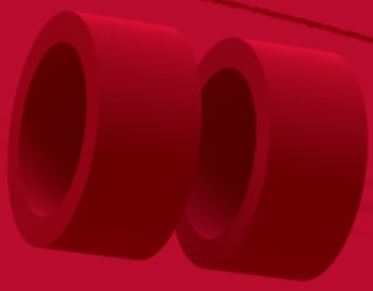
# RBR SENSORS

# COMMAND REFERENCE

RBR*coda*$^3$ RBR*tridente*

RBR*coda* RBR*quadrante*



rbr-global.com

# Table of contents


**1 Sensor command reference ..... 4**

**2 Quick start ..... 5**

2.1 Streaming ..... 5

2.2 Polling ..... 5

2.3 Channels introspection ..... 6

**3 Channel types ..... 7**

**4 Channel labels ..... 9**

**5 Realtime data format ..... 16**

**6 Command entry ..... 18**

**7 Error messages ..... 19**

**8 Commands ..... 20**

8.1 sampling ..... 20

8.2 outputformat ..... 23

8.3 stream ..... 26

8.4 fetch ..... 26

8.5 calibration ..... 28

8.6 channel ..... 30

8.7 sensor ..... 33

8.8 channels ..... 34

8.9 settings ..... 36

8.10 serial ..... 37

8.11 getall ..... 38
8.12 id ..... 39
8.13 info ..... 40
8.14 confirmation ..... 40
8.15 prompt ..... 41
8.16 config ..... 42
**9 Revision history ..... 43**

# 1 Sensor command reference

Sensor commands use the following formatting conventions:

1. Examples of literal input to and output from the sensor are shown in `bold` type.
2. In examples of dialogue between the sensor and a host, input to the sensor is preceded by `>>`, while output from the sensor is preceded by `<<`. These characters must not actually be included in commands or expected in responses.
3. Some examples of command dialogues contain descriptive comments which are not part of the command or response. These start with a percent character, `%`.
4. When an item or group of items is optional, it is enclosed in `[square brackets]`.
5. Where an item can be only one of several options, options are separated by vertical bars, `|`.
6. Placeholders for variable fields are in `<enclosed in angle brackets>`.
7. Lists are used for unknown or variable numbers of items, or to abbreviate large numbers of options. These are specified by giving a first example of an item, followed by a comma and ellipsis: `<example-value>, … `.

# 2 Quick start

You must have a serial terminal to set up your realtime sensor in the polled mode.

## 2.1 Streaming

When shipped out of the factory, the RBR sensors (RBR*coda,* RBR*coda*$^3$, RBR*tridente*, RBR*quadrante*) are configured to stream upon power cycle at 9600 baud and the maximum possible sampling rate.

The maximum sampling rate for RBR*coda* T.ODO is 1Hz

It is possible to adjust the serial baudrate and sampling rate via the **serial** and sampling commands respectively. Use the stream command to turn streaming mode on and off.

Here is an example of how to configure/enable streaming on a sensor (RBR*tridente*)

Ensure the serial baudrate is correct:

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

Set the sampling rate to 8Hz:

```
>> sampling period = 125
<< sampling period = 125
```

Enable streaming:

```
>> stream state = on
<< stream state = on
```

After you issue the stream command, the sensor will start streaming:

```
<< 0, 2.6534132, 22.0217241, 1.9596633
<< 125, 2.6564438, 22.0242156, 1.9542156
<< 500, 2.6574234, 22.0278541, 1.9575842
...
<< 10000, 2.6534485, 22.0296523, 1.9514527
```

## 2.2 Polling

It is possible to operate an RBR sensor in polled mode. Use the stream command to disable the streaming mode (default when shipped out of the factory), then use the fetch command to request one sample. For example, with an RBR*tridente*:

Stop streaming:

```
>> stream state = off
<< stream state = off
```

Poll a sample:.

```
>> fetch
<< 500, 2.6574234, 22.0278541, 1.9575842
```

## 2.3 Channels introspection

Use the channels and **channel** commands to determine the list of channels available and their type (see also Channels types and Channel labels).

For example, with an RBR*tridente* bb.chl-a.fDOM:

```
>> channels count
<< channels count = 3
>> channel 1 label, type
<< channel 1 label = backscatter_00, type = turb21
>> channel 2 label, type
<< channel 2 label = chlorophyll_00, type = fluo48
>> channel 3 label, type
<< channel 3 label = fdom_00, type = fluo46
```

Use the **calibration** command sent to the RBR sensor to retrieve each channel's calibration:

```
>> calibration fdom_00
<< calibration fdom_00 index = 3, datetime = 20201204213302, c0 = 3.3878490e-003, c1 =
-275.76637e-006
```

Please note the number of coefficients might change depending on the channel.

Furthermore, for some channels, the **sensor** command retrieves additional information. In particular, use this command to identify the wavelength of each radiometer channel on the RBR*quadrante*:

```
>> sensor irradiance_01
<< sensor irradiance_01  wavelength = 445/10nm
```

In a similar way, retrieve additional information on the RBR*tridente* channels:

```
>> sensor fluo_45
<< sensor chlorophyll_00 chla_00_median = , chla_00_stdev =
```

# 3 Channel types

Channel types are short, pre-defined names for the installed channels. These names are used by the channel command.

**Sensor types used in continuous mode**

| Channel type | Description |
|---|---|
| `doxy26` | Dissolved oxygen saturation, RBR*coda* DO (OxyGuard) |
| `doxy21, doxy24` | Dissolved oxygen concentration, RBR*coda* T.ODO |
| `doxy22` | Dissolved oxygen saturation, derived, RBR*coda* T.ODO |
| `fluo45, fluo48, fluo49` | Chlorophyll *a*, RBR*tridente* |
| `fluo46` | fDOM, RBR*tridente* |
| `fluo50` | Rhodamine, RBR*tridente* |
| `fluo52` | Phycocyanin, RBR*tridente* |
| `fluo54` | Phycoerythrin, RBR*tridente* |
| `fluo63` | Chlorophyll, RBR*coda* chl-a |
| `irr_03` | Narrow-band irradiance, RBR*coda* rad |
| `irr_07` | Narrow-band irradiance, RBR*quadrante* |
| `opt_05` | Phase, RBR*coda* T.ODO |
| `par_02` | Photosynthetically active radiation, RBR*coda* PAR (LI-COR) |
| `par_04` | Photosynthetically active radiation, RBR*coda* PAR |
| `par_07` | Photosynthetically active radiation, RBR*quadrante* |
| `pres25` | Pressure, with temperature correction, RBR*coda*$^3$ D |
| `pres26` | Pressure, with temperature correction, RBR*coda*$^3$ T.D |
| `temp18, temp21` | Temperature, RBR*coda*$^3$ T |
| `temp12, temp20` | Temperature, RBR*coda*$^3$ T.D |

| Channel type | Description |
|---|---|
| `temp15` | Temperature, RBR*coda* T.ODO |
| `turb05` | Turbidity, RBR*coda* Tu (Seapoint) |
| `turb15` | Turbidity, RBR*tridente* |
| `turb16, turb21` | Backscatter, RBR*tridente* |
| `turb17, turb18` | Turbidity, RBR*coda* Tu |

**Sensor types used in aggregate mode**

| Channel type | Description |
|---|---|
| `mdn_00` | Median value of the parent parameter |
| `sd__00` | Standard deviation value of the parent parameter |

# 4 Channel labels

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

Instruments with the aggregate mode also support statistics channels:

- channels holding the median of the aggregate are suffixed with `_median`
- channels holding the standard deviation of the aggregate are suffixed with `_std`

| Sensor | Channel type | Label |
|---|---|---|
| RBR*coda*$^3$ T | `temp18` | `temperature_00` |
| RBR*coda*$^3$ T | `temp21` | `temperature_00` |
| RBR*coda*$^3$ D | `pres25` | `pressure_00` |
| RBR*coda*$^3$ T.D | `temp12`<br>`pres26` | `temperature_00`<br>`pressure_00` |
| RBR*coda*$^3$ T.D | `temp20`<br>`pres26` | `temperature_00`<br>`pressure_00` |
| RBR*coda* DO (OxyGuard) | `doxy26` | `oxygensaturation_00` |
| RBR*coda* T.ODO | `temp15`<br>`doxy21`<br>`doxy22`<br>`doxy24`<br>`opt_05` | `odotemperature_00`<br>`oxygenconcentration_00`<br>`oxygensaturation_00`<br>`oxygenconcentration_01`<br>`odophase_00` |
| RBR*coda* rad | `irr_03` | `irradiance_00` |
| RBR*coda* PAR | `par_04` | `par_00` |
| RBR*coda* PAR (LI-COR) | `par_02` | `par_00` |
| RBR*coda* Tu (Seapoint) | `turb05` | `turbidity_00` |

| Sensor | Channel type | Label |
|---|---|---|
| RBR*coda* Tu | `turb17`<br>`turb18`<br>`mnd_00`<br>`sd__00`<br>`mnd_00`<br>`sd__00` | `turbidity_00`<br>`turbidity_obs_00`<br>`turbidity_00_median`<br>`turbidity_00_std`<br>`turbidity_obs_00_median`<br>`turbidity_obs_00_std` |
| RBR*coda* chl-*a* | `fluo63` | `chlorophyll_00` |
| RBR*tridente* bb.bb.bb | `turb21`<br>`turb21`<br>`turb21`<br>`mdn_00`<br>`sd__00`<br>`mdn_00`<br>`sd__00`<br>`mnd_00`<br>`sd__00` | `backscatter_00`<br>`backscatter_01`<br>`backscatter_02`<br>`backscatter_00_median`<br>`backscatter_00_std`<br>`backscatter_01_median`<br>`backscatter_01_std`<br>`backscatter_02_median`<br>`backscatter_02_std` |
| RBR*tridente* bb.bb.chl-a | `turb16`<br>`turb16`<br>`fluo45`<br>`mdn_00`<br>`sd__00`<br>`mdn_00`<br>`sd__00`<br>`mnd_00`<br>`sd__00` | `backscatter_00`<br>`backscatter_01`<br>`chlorophyll_00`<br>`backscatter_00_median`<br>`backscatter_00_std`<br>`backscatter_01_median`<br>`backscatter_01_std`<br>`chlorophyll_00_median`<br>`chlorophyll_00_std` |

| Sensor | Channel type | Label |
|---|---|---|
| RBR*tridente* bb.bb.chl-a | `turb21`<br>`turb21`<br>`fluo48`<br>`mdn_00`<br>`sd__00`<br>`mdn_00`<br>`sd__00`<br>`mnd_00`<br>`sd__00` | `backscatter_00`<br>`backscatter_01`<br>`chlorophyll_00`<br>`backscatter_00_median`<br>`backscatter_00_std`<br>`backscatter_01_median`<br>`backscatter_01_std`<br>`chlorophyll_00_median`<br>`chlorophyll_00_std` |
| RBR*tridente* bb.bb.chl-a | `turb21`<br>`turb21`<br>`fluo49`<br>`mdn_00`<br>`sd__00`<br>`mdn_00`<br>`sd__00`<br>`mnd_00`<br>`sd__00` | `backscatter_00`<br>`backscatter_01`<br>`chlorophyll_00`<br>`backscatter_00_median`<br>`backscatter_00_std`<br>`backscatter_01_median`<br>`backscatter_01_std`<br>`chlorophyll_00_median`<br>`chlorophyll_00_std` |
| RBR*tridente* bb.chl-a.chl-a | `turb21`<br>`fluo49`<br>`fluo49`<br>`mdn_00`<br>`sd__00`<br>`mdn_00`<br>`sd__00`<br>`mnd_00`<br>`sd__00` | `backscatter_00`<br>`chlorophyll_00`<br>`chlorophyll_01`<br>`backscatter_00_median`<br>`backscatter_00_std`<br>`chlorophyll_00_median`<br>`chlorophyll_00_std`<br>`chlorophyll_01_median`<br>`chlorophyll_01_std` |

| Sensor | Channel type | Label |
|---|---|---|
| RBR*tridente* bb.chl-a.fDOM | `turb16`<br>`fluo45`<br>`fluo46`<br>`mdn_00`<br>`sd__00`<br>`mdn_00`<br>`sd__00`<br>`mnd_00`<br>`sd__00` | `backscatter_00`<br>`chlorophyll_00`<br>`fdom_00`<br>`backscatter_00_median`<br>`backscatter_00_std`<br>`chlorophyll_00_median`<br>`chlorophyll_00_std`<br>`fdom_00_median`<br>`fdom_00_std` |
| RBR*tridente* bb.chl-a.fDOM | `turb21`<br>`fluo48`<br>`fluo46`<br>`mdn_00`<br>`sd__00`<br>`mdn_00`<br>`sd__00`<br>`mnd_00`<br>`sd__00` | `backscatter_00`<br>`chlorophyll_00`<br>`fdom_00`<br>`backscatter_00_median`<br>`backscatter_00_std`<br>`chlorophyll_00_median`<br>`chlorophyll_00_std`<br>`fdom_00_median`<br>`fdom_00_std` |
| RBR*tridente* chl-a.chl-a.phycoc | `fluo48`<br>`fluo48`<br>`fluo52`<br>`mdn_00`<br>`sd__00`<br>`mdn_00`<br>`sd__00`<br>`mnd_00`<br>`sd__00` | `chlorophyll_00`<br>`chlorophyll_01`<br>`phycocyanin_00`<br>`chlorophyll_00_median`<br>`chlorophyll_00` _std<br>`chlorophyll_01_median`<br>`chlorophyll_01` _std<br>`phycocyanin_00_median`<br>`phycocyanin_00` _std |

| Sensor | Channel type | Label |
|---|---|---|
| RBR*tridente* chl-a.fDOM.Tu | `fluo45`<br>`fluo46`<br>`turb15`<br>`mdn_00`<br>`sd__00`<br>`mdn_00`<br>`sd__00`<br>`mnd_00`<br>`sd__00` | `chlorophyll_00`<br>`fdom_00`<br>`turbidity_00`<br>`chlorophyll_00_median`<br>`chlorophyll_00_std`<br>`fdom_00_median`<br>`fdom_00_std`<br>`turbidity_00_median`<br>`turbidity_00_std` |
| RBR*tridente* chl-a.fDOM.Tu | `fluo48`<br>`fluo46`<br>`turb15`<br>`mdn_00`<br>`sd__00`<br>`mdn_00`<br>`sd__00`<br>`mnd_00`<br>`sd__00` | `chlorophyll_00`<br>`fdom_00`<br>`turbidity_00`<br>`chlorophyll_00_median`<br>`chlorophyll_00_std`<br>`fdom_00_median`<br>`fdom_00_std`<br>`turbidity_00_median`<br>`turbidity_00_std` |
| RBR*tridente* chl-a.phycoc.Tu | `fluo48`<br>`fluo52`<br>`turb15`<br>`mdn_00`<br>`sd__00`<br>`mdn_00`<br>`sd__00`<br>`mnd_00`<br>`sd__00` | `chlorophyll_00`<br>`phycocyanin_00`<br>`turbidity_00`<br>`chlorophyll_00_median`<br>`chlorophyll_00_std`<br>`phycocyanin_00_median`<br>`phycocyanin_00` _std<br>`turbidity_00_median`<br>`turbidity_00_std` |

| Sensor | Channel type | Label |
|---|---|---|
| RBR*tridente* fDOM.phycoc.phycoe | `fluo46`<br>`fluo52`<br>`fluo54`<br>`mdn_00`<br>`sd__00`<br>`mdn_00`<br>`sd__00`<br>`mnd_00`<br>`sd__00` | `fdom_00`<br>`phycocyanin_00`<br>`phycoerythrin_00`<br>`fdom_00_median`<br>`fdom_00_std`<br>`phycocyanin_00_median`<br>`phycocyanin_00` _std<br>`phycoerythrin_00_median`<br>`phycoerythrin_00_std` |
| RBR*tridente* phycoc.phycoe.rhod | `fluo52`<br>`fluo54`<br>`fluo50`<br>`mdn_00`<br>`sd__00`<br>`mdn_00`<br>`sd__00`<br>`mnd_00`<br>`sd__00` | `phycocyanin_00`<br>`phycoerythrin_00`<br>`rhodamine_00`<br>`phycocyanin_00_median`<br>`phycocyanin_00` _std<br>`phycoerythrin_00_median`<br>`phycoerythrin_00_std`<br>`rhodamine_00_median`<br>`rhodamine_00_std` |
| RBR*quadrante* irr.irr.irr.irr | `irr_07`<br>`irr_07`<br>`irr_07`<br>`irr_07`<br>`mdn_00`<br>`sd__00`<br>`mdn_00`<br>`sd__00`<br>`mnd_00`<br>`sd__00`<br>`mnd_00`<br>`sd__00` | `irradiance_00`<br>`irradiance_01`<br>`irradiance_02`<br>`irradiance_03`<br>`irradiance_00_median`<br>`irradiance_00_std`<br>`irradiance_01_median`<br>`irradiance_01_std`<br>`irradiance_02_median`<br>`irradiance_02_std`<br>`irradiance_03_median`<br>`irradiance_03_std` |

| Sensor | Channel type | Label |
|---|---|---|
| RBR*quadrante* irr.irr.irr.PAR | `irr_07`<br>`irr_07`<br>`irr_07`<br>`par_07`<br>`mdn_00`<br>`sd__00`<br>`mdn_00`<br>`sd__00`<br>`mnd_00`<br>`sd__00`<br>`mnd_00`<br>`sd__00` | `irradiance_00`<br>`irradiance_01`<br>`irradiance_02`<br>`par_00`<br>`irradiance_00_median`<br>`irradiance_00_std`<br>`irradiance_01_median`<br>`irradiance_01_std`<br>`irradiance_02_median`<br>`irradiance_02_std`<br>`par_00_median`<br>`par_00_std` |

# 5 Realtime data format

When stream is set to `on`, sensors will begin streaming data as soon as power is provided. The time to the first sample depends on the sensor.

> The time to the first sample for the RBR*quadrante* and the RBR*tridente* is <100ms.

Below are example strings of data from each variety of the RBR sensors:

| Instrument | Example data | Channels list |
|---|---|---|
| RBR*coda* T.ODO | `29000, 23.2868, 200.4000, 93.0000, 245.0000, 29.6900` | Timestamp (ms), Temperature (°C), concentration compensated for salinity (µmol/L), saturation (%), concentration uncompensated for salinity (µmol/L), phase (°) |
| RBR*coda*$^3$ T | `29000, 23.2868` | Timestamp (ms), Temperature (°C) |
| RBR*coda*$^3$ D | `29000, 10.2484` | Timestamp (ms), Pressure (dbar) |
| RBR*coda* DO | `29000, 98.8754` | Timestamp (ms), Saturation (%) |
| RBR*coda*$^3$ T.D | `29000, 23.2868, 10.2484` | Timestamp (ms), Temperature (°C), Pressure (dbar) |
| RBR*coda* PAR, RBR*quadrante* | `1250, 38.6671142` | Timestamp (ms), PAR (µmol/m$^2$/s) |
| RBR*coda* rad, RBR*quadrante* | `1250, 22.0217241` | Timestamp (ms), Irradiance (µW/cm$^2$/nm) |
| RBR*tridente* | `3550, 0.0008541, 22.0217241, 1.9596633` | Timestamp (ms), backscatter (m$^{-1}$sr$^{-1}$), chlorophyll *a* (µg/L), fDOM (ppb) |
| RBR*coda* Tu | `29000, 35.6674542` | Timestamp (ms), Turbidity (FTU) |
| RBR*coda* chl-*a* | `29000, 5` | Timestamp (ms), Chlorophyll (µg/L) |

The timestamp is the number of milliseconds since the time of the first sample, and so the first sample is always timestamped as zero. The timestamp will restart any time the CPU resets or if the following sample parameters are changed:

- Changing the sampling period
- Changing the sampling mode
- Changing the burst length or burst interval
- Turning a channel on/off (advanced users only)

The values following the timestamp are all the enabled channels. To identify which channels are enabled and their order in the string you can use the command **outputformat channelslist**.

If a channel reports an error, it will report `Error-<EC>` instead of measured values. All other channels will continue to report valid data.
See the outputformat section for the list of error codes.

**Example**

```
>> outputformat channelslist
<< outputformat channelslist = temperature (C), O2_concentration (umol/L),
O2_air_saturation (%), uncompensated_O2_concentration (umol/L)
```

# 6 Command entry

**Start and end of a command**

A *potential* command is considered to begin when its first character is received. 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's prompt, and a total of two prompts are sent (see also the prompt command).

**Uppercase and lowercase**

In general, the sensor 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. The case of the output sensor response may or may not match the case of the input.

**Common error messages**

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

1. `E0102 invalid command  <unknown-command-name>`
2. `E0107 expected argument missing`
3. `E0108 invalid argument to command: <invalid-argument>`

See Error messages .

# 7 Error messages

Refer to the list below for the most common error messages.

Each error message begins with *Ennnn*, where *nnnn* is a four-digit decimal number, padded with leading zeroes if necessary.

The number allows host software to interpret the error code correctly even if the rather terse messages from the sensor are unsuitable for any reason.

Note that some messages may be followed by variable elements not shown here.

| Key | Description |
|---|---|
| `E0102` | `invalid command <unknown-command-name>` |
| `E0107` | `expected argument missing` |
| `E0108` | `invalid argument to command: <invalid-argument>` |
| `E0109` | `feature not available` |
| `E0111` | `command failed` |
| `E0114` | `feature not supported by hardware` |
| `E0410` | `no sampling channels active` |
| `E0501` | `item is not configured` |
| `E0505` | `no channels configured` |

You may encounter errors not included in the list above. In that case, please contact RBR for help.

# 8 Commands

## 8.1 sampling

**Usage**

`>> sampling [ mode | period | availablefastperiods | userperiodlimit | aggregate | measurementperiod | measurementcount | burstinterval | burstlength]`

- The RBR*coda* T.ODO has a max sample rate of 1Hz (period = 1000)
- The RBR*tridente* does not support `period` values < 1000 when in `aggregate` mode

**Description**

Allows various parameters to be reported.

**Parameters**

- `mode [= <mode>]` reports the sampling mode for the current schedule:
  - `continuous`

    Measurements are taken at the specified `period` between the start and end times.
    - The following parameters are required to be configured for this sampling mode:
      - `period [= <period>]` reports or sets the output sample period in milliseconds. All values between 1000 and 86400000 in multiple of 1000, and all values reported by `availablefastperiods` are acceptable.
  - `aggregate` – **Supported by**: RBR*quadrante* (firmware v1.004 and up), RBR*tridente* (firmware v1.017 and up: *see info box at top of section for exceptions*), and RBR*coda* Tu (firmware v1.003 and up).
    Measurement bursts are taken at the period indicated by the `period` parameter. The `measurementperiod` parameter is the effective sampling period within the burst, and the parameter `measurementcount` is the number of measurements within the burst. Statistical channels (channels holding the median or the standard deviation of the aggregate) will be calculated using all the readings from the burst, and the non-statistical channel will report the arithmetic mean. See Channel labels.
    - The following parameters are required to be configured for this sampling mode:
      - `measurementperiod [= <measurementperiod>]` reports or sets the instrument's effective internal sampling rate in milliseconds. All values between 1000 and 86400000 in multiple of 1000, and all values reported by `availablefastperiods` are acceptable.
      - `measurementcount [= <measurementcount>]` reports or sets the number of measurements to be taken in a burst.
      - `period [= <period>]` reports or sets the output sample period in milliseconds.
  - `tide` – **Supported by**: instruments which are so configured in the factory.
    Measurements are taken in bursts between the start and end times. The average value of all measurements in a burst is computed and stored as a single sample value at the end of the burst. The time between the start of two consecutive bursts is given by the `burstinterval`, the number of measurements in the burst is given by the `burstlength`, and the time between measurements within

the burst is given by the `period`. This mode allows host software to process the data for tide monitoring applications.
    - The following parameters are required to be configured for this sampling mode:
        - `period [= <period>]` reports or sets the instrument's effective internal sampling rate in milliseconds. All values between 1000 and 86400000 in multiple of 1000, and all values reported by `availablefastperiods` are acceptable.
        - `burstlength [= <burstlength>]` reports or sets the number of measurements taken in each burst. The permitted range is 2 to 65535, but before the instrument can be enabled for sampling the value set must also be consistent with the measurement `period` and `burstinterval`.
        - `burstinterval [= <burstinterval>]` reports or sets the output sample period in milliseconds. It is the time between the first measurement of two consecutive bursts; it is not the gap between the end of one burst and the start of the next.
- `period [= <period>]` reports or sets the output sample period in milliseconds.
    - In `continuous` mode, this is the instrument's effective output sampling rate. All values between 1000 and 86400000 in multiple of 1000, and all values reported by `availablefastperiods` are acceptable.
    - In `aggregate` mode, reports or sets this is the instrument's effective output sampling rate, i.e. the period between the start of two consecutive bursts. All values between 1000 and 86400000 in multiple of 1000, and all values reported by `availablefastperiods` are acceptable.
    - In `tide` mode, this is the time between continuous measurements *within* the burst.
    All values between 1000 and 255000 in multiple of 1000, and all values reported by `availablefastperiods` are acceptable.
- `availablefastperiods` reports a list of the fast measurement periods available for 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, `|`. If there are no fast periods available, the word `none` is returned instead of a list of values. When a list is reported, the `period` for sampling rates faster than 1Hz can be set to any value in the list, but no others.
Note that the periods on the list may be supported in some modes but not others, depending on the instrument's configuration.
- `userperiodlimit` reports the minimum period which can be used in 'fast' sampling modes, in milliseconds; the `period` can not be set to a value less than this.

The following parameters are available only if the sensor supports `aggregate` mode.

- `measurementperiod [= <measurementperiod>]` is the instrument's effective internal sampling rate. All values between 1000 and 86400000 in multiple of 1000, and all values reported by `availablefastperiods` are acceptable.
- `measurementcount [= <measurementcount>]` is the number of measurements to be taken in a burst.



In aggregate mode, the constraining relationship between sampling parameters is: `period >= (measurementcount * measurementperiod), and period >= 1000.`

The following parameters are available only if the sensor supports `tide` mode.

- `burstinterval [= <burstinterval>]` reports or sets the time between the first measurement of two consecutive bursts; it is not the gap between the end of one burst and the start of the next. The <*burstinterval*> is specified in milliseconds.
  The absolute limits of the permitted range are 1000 (one second) to 86400000 (24 hours), and the <*burstinterval*> must be set to a multiple of 1000. However, before the instrument can be enabled for sampling the value set must also be consistent with the measurement `period` and `burstlength`.
- `burstlength [= <burstlength>]` reports or sets the number of measurements taken in each burst. The permitted range is 2 to 65535, but before the instrument can be enabled for sampling the value set must also be consistent with the measurement `period` and `burstinterval`.

⚠ In tide mode, the constraining relationship between sampling parameters is: `burstinterval > (burstlength * period)`

**Examples**

```
>> sampling
<< sampling mode = continuous, period = 1000, burstlength = 10, burstinterval = 300000,
userperiodlimit = 1000
```

RBR*coda* T.ODO has a max sample rate of 1Hz.

```
>> sampling
<< sampling mode = continuous, period = 125, userperiodlimit = 63
```

The instrument has been programmed for continuous 8Hz sampling. This instrument does not support any of the other modes, so the parameters are not reported.

```
>> sampling
<< sampling mode = continuous, period = 63, userperiodlimit = 63, burstlength = 60,
burstinterval = 300000
```

The instrument has been programmed for continuous 16Hz sampling. The programmed values of the burst parameters are reported but do not apply to **continuous** mode.

```
>> sampling mode = tide
<< sampling mode = tide
>> sampling
<< sampling mode = tide, period = 63, userperiodlimit = 63, burstlength = 60,
burstinterval = 300000
```

**Tide** mode is enabled without changing the other parameters; the instrument is now programmed to take a burst of 60 measurements at 16Hz every five minutes and store the average of the 60 measurements.

```
>> sampling mode
<< sampling mode = aggregate
>> sampling
<< sampling mode = aggregate, period = 2000, userperiodlimit = 63, measurementcount = 8,
measurementperiod = 250
```

**Aggregate** mode is enabled without changing the other parameters; the instrument is now programmed to take a burst of 8 measurements at 4Hz every 2 seconds and store the statistics of the 8 measurements.

```
>> sampling availablefastperiods
<< sampling availablefastperiods = 500|125|63
>> sampling availablefastperiods
<< sampling availablefastperiods = none
```

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

**Errors**

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

The command was given with an argument which is unrecognised or misplaced.

`Error E0109 feature not available`

An attempt was made to use a feature which the instrument is not configured to support; for example "sampling mode = tide" if the instrument does not support tidal averaging.

## 8.2 outputformat

**Usage**

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

For the RBR*coda* T.ODO:

`>> outputformat [ type | channelslist ]`

The RBR*coda* T.ODO only supports a single `outputformat` : `caltext06`

**Description**

Reports or sets the format used to transmit data over the communications link.

This format applies to both fetched data and live streamed data, if available. If no arguments are given, the current setting of the `type` parameter is reported.

**Parameters**

- `type [= <type>]` sets and/or reports the current output format type. Not all instruments support all the formats; query the `availabletypes` parameter to check which formats are available.
- `availabletypes` reports the formats which are available. **Not supported by**: RBR*coda* T.ODO.
  The supported formats:
  - `caltext06` , calibrated ASCII output.
    The output starts with a duration (milliseconds) since power up. A value for each channel is then sent; this is the measured parameter after conversion to physical units according to the instrument's current calibration. All values are shown to four decimal places. A comma and space separate the timestamp and values. The timestamp is in milliseconds.
    Format: `timestamp, <value1>, <value2>, ...`
    Examples for a two-channel instrument: `10000, 38.6664, 21.5183`
  - `caltext08` , calibrated ASCII output.
    The output starts with a duration (milliseconds) since power up. A value for each channel is then sent; this is the measured parameter after conversion to physical units according to the instrument's current calibration. All values are shown to nine significant digits, ensuring no loss of resolution. A comma and space separate the timestamp and values. The timestamp is in milliseconds.
    Format: `timestamp, <value1>, <value2>, ...`
    Examples for PAR and rad sensors: `1250, 38.6671142, 22.0217241, 10.9596633`
- `channelslist` reports a list of names and units for the active channels, in order. **Supported by:** RBR*coda* T.ODO, RBR*coda*$^3$ T, RBR*coda*$^3$ D, RBR*coda* DO, RBR*coda*$^3$ T.D, RBR*coda* PAR, RBR*coda* rad.
  This list is helpful in identifying the channel corresponding to each value in the transmitted data. Any channels which have been turned `off` are excluded. The list expands with support added for more sensor types over time, so not all firmware versions will support every sensor type listed. If there is more than one channel of a particular type, the same information is reported for all of them. The list of all possible names is given below. Some are quite generic, and others are very sensor-specific :
  - `amplitude`
  - `depth`
  - `dissolved-O2`
  - `irradiance`
  - `O2_air_saturation`
  - `O2_concentration`
  - `uncompensated_O2_concentration`
  - `phase`
  - `PAR`
  - `period`
  - `pressure`
  - `temperature`
  - `turbidity`
  - `voltage`

  Note this command responds differently for the following sensors. RBR*quadrante,* RBR*tridente,* RBR*coda*

Tu, and RBR*coda* chl-*a.* These sensors report the channel types rather than channel names.
**Example**

```
outputformat channelslist = fluo63
```

- `labelslist` reports the list of active channel labels, in order. **Not supported by**: RBR*coda* T.ODO.
  This list is helpful in identifying the channel corresponding to each value in the transmitted data. Any channels which have been turned **off** are excluded. A channel's label is set upon request for OEM customers (see the **channel** command).
  Note that any individual channel value may be replaced by one of the following:
  - `Error-` < `EC` >: an identifiable error occurred on this channel, represented by one of the following two-digit error codes:
    00 - generic or unknown error
    09 - no sample was started
    10 - sample acquisition still in progress
    11 - sample process failed somewhere
    14 - supporting channel value not valid, or unknown equation
    16 - channel value is outside reasonable range
    21 - channel is not correctly calibrated
- `nan` : the value is Not A Number in IEEE floating point format, which indicates an internal problem with calculating the value
- `inf / -inf` : attempting to calculate the value produced a result outside the range which can be represented
- `###` : the channel is not calibrated, so an output value could not be calculated

**Examples**

```
>> outputformat
<< outputformat type = caltext06, labelslist = temperature_00|pressure_00
```

```
>> outputformat type
<< outputformat type = caltext06
```

```
>> outputformat availabletypes
<< outputformat availabletypes = caltext06|caltext08
```

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

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

**Errors**

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

The command was given with an argument which is unrecognised or misplaced.

## 8.3 stream

**Usage**

`>>stream [state]`

**Description**

Turns data streaming on or off.

If the command is given with no parameters, the value of the `state` parameter is reported. The operating parameters of the serial link, such as baudrate and mode, are accessed using the **serial** command.

- `state [= on | off]` reports the state of the streamed data feature, and optionally turns it on or off. When the feature is on, acquired data is sent over the serial link. When the feature is off, data are not sent.

**Examples**

```
>> stream
<< stream state = off
```

```
>> stream state = on
<< stream state = on
```

**Errors**

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

The command was given with an argument which is unrecognised or misplaced.

## 8.4 fetch

**Usage**

`>> fetch [ channels ]`

For RBR*coda* T.ODO:

`>> fetch`

RBR*coda* T.ODO ignores the `[ channels ]` argument and will always output data for all channels.

**Description**

Requests an "on-demand" sample set from the sensor.

If a recent scheduled sample set is available, those values may be returned to satisfy the `fetch` request. "Recent" in this context means less than 500ms old. Otherwise, a sample set is explicitly acquired for the benefit of the `fetch`.

The sensor simply responds with the `<sample-data>`. Depending on the configured settling time (or power-on settling delay) for the attached sensors, there may be a noticeable delay before the `<sample-data>` appears. Refer to the channels command for further discussion of settling time.

**Parameters**

- `channels = <listofchannels>`
  The list of channels to be acquired, using either indices or labels of channels (see channel). At least one valid channel must be specified. **Not supported by**: RBR*coda* T.ODO.
  Without `channels = <listofchannels>`, the `fetch` command reports `<sample-data>` with the channels in the expected order, as revealed by either outputformat `channelslist` or outputformat `labelslist`. If the `<listofchannels>` specifies the channels in a different order than this default, then they are reported in the same order as given by the `<listofchannels>`.

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

**Examples**

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

```
>> fetch
<< 29000, 23.2868, 10.2484
```

The instrument will fetch a sample from all channels.

```
>> channel 1 type
<< channel 1 type = temp12
>> channel 2 type
<< channel 2 type = pres26
>> >> outputformat
<< outputformat type = caltext06, labelslist = temperature_00|pressure_00
>> fetch
<< 29000, 23.2868, 10.2484
>> fetch channels = 2|1
<< 31000, 23.2870, 10.2502
>> fetch channels = pressure_00
<< 35000, 10.2495
```

The instrument will fetch a sample from specific channels only.

**Errors**

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

The command was given with an argument which is unrecognised or misplaced.

`Error E0410 no sampling channels active`

The instrument has no channels activated for sampling.

## 8.5 calibration

**Usage**

`>> calibration <indexorlabel> [ datetime | c0 | ... | c<Nc> | x0 | ... | x<Nx> | n0 | ... | n<Nn> ]`

For RBR*coda* T.ODO:

`>> calibration <index> [ datetime | c0 | ... | c<Nc> | x0 | ... | x<Nx> | n0 | ... | n<Nn> ]`

**Description**

Reports information regarding the most recent calibration for the channel specified by `<indexorlabel>.`

**Parameters**

`<indexorlabel>` is a required parameter in all cases. The RBR*coda* T.ODO *only* supports `<index>`.

The number and types of coefficients reported will vary depending on the channel `type` (see channel for more information).

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, and the `calibration` command helps to manage that information.

If given with only the `<indexorlabel>`, the command reports all information applicable to the channel. Requesting just the `<label>` or just the `<index>` will return an error message (E0108); it must always be `<indexorlabel>.` However, you can request `datetime` on its own.

Coefficients in each group may be requested all together by using the group name: `c`, `x`, or `n`. Requesting an item which does not exist (eg. **c3** for a linear sensor) may result in either an error message, or a response such as `c3 = n/a`.

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

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

Additionally, `datetime` is reported using a <*YYYYMMDDhhmmss*> format. It is the date and time of the most recent calibration change for the channel.
Coefficients are arranged in three groups, `c0…`, `x0…`, and there is a further group `n0…` of cross-channel reference indices. The purpose and function of each are described below. The groups may be referred to by name: `c`, `x` or `n`.

- `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.
  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 reported for only some equation types, namely those which employ cross-channel compensation or correction of the primary value using one or more inputs from other channels in the logger. `x0, x1...` are also coefficient values which follow the same rules as the **c** group. The exact function of each coefficient depends on the equation used.
- `n0, n1…` apply only to some equation types, those using cross-channel compensation or correction. They are not coefficients, but (in general) the indices of other instrument channels whose data are also inputs to the equation for channel < `indexorlabel` >. This permits output data to depend on more than one channel; for example, to be corrected for temperature dependencies. The values of **n0, n1,** etc. are simple integer numbers. Remember that the index of the first channel is `1`; zero is not valid.

Equations which use the `x0, x1…` coefficients require at least one `n` index. The instrument may also have "derived parameter" channels, which are output values computed from measured channels: a good example would be pressure, which requires correction by a separately measured temperature channel.. In such cases `n0` is required to tell the instrument to use the input from the supporting channel.

When setting new values in either the `c` or the `x` group of coefficients, two rules apply.

1. A value for `datetime = <YYYYMMDDhhmmss>` *must* also be specified, as the `datetime` must reflect the most recent coefficient change.
2. Only one group at a time can be accessed. You cannot set both `c` and `x` coefficient values in a single command. Within either group, one, all, or any number of the coefficients in between may be set.

⚠ Any value for any **n** coefficient can be set only at the factory.

There is one special case when the value of an `n` index may be the text field " `value` ". Like all other **n** coefficient settings, this can be set only 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 settings command will be used. An example would be the derived channel for depth, which depends on measured pressure but also requires a value for atmospheric pressure, available via the settings command.

**Examples**

```
>> calibration 2
<< calibration 2 label = pressuretemperature_00, datetime = 20201204213302, c0 =
3.3878490e-003, c1 = -275.76637e-006, c2 = 3.9983520e-006, c3 = -221.88040e-009
```

```
>> calibration pressuretemperature_00
<< calibration pressuretemperature_00 index = 2, datetime = 20201204213302, c0 =
3.3878490e-003, c1 = -275.76637e-006, c2 = 3.9983520e-006, c3 = -221.88040e-009
```

The instrument is responding with the calibration for a single channel by index and by label.

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

The instrument is responding with the calibration for all channels by indices and by labels.

```
>>calibration 1 datetime = 20171203134201, c0 = 3.5000000e-003, c1 = -250.00002e-006, c2
= 2.7000000e-006, c3 = 23.000000e-009
<<calibration 1 label = temperature_00, datetime = 20000401000000, c0 = 3.5000000e-003,
c1 = -250.00002e-006, c2 = 2.7000000e-006, c3 = 23.000000e-009
```

The instrument is setting the calibration coefficients and time.

**Errors**

`Error E0107 expected argument missing`

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

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

The command was given with an argument which is unrecognised or misplaced, for example:

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

`Error E0501 item is not configured`

There is a problem with the configuration of the specified channel. Please contact RBR for help.

`Error E0505 no channels configured`

There is a serious fault with the instrument. Please contact RBR for help.

## 8.6 channel

**Usage**

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

For the RBR*coda* T.ODO:

`>> channel <index> [ type | module | status | latency | readtime | equation | userunits ]`

**Description**

Returns information about the channel at the specified <*indexorlabel*>.

**Parameters**

Channels can be identified either by an index (an integer) or by their label (a meaningful string, for example temperature_00). The first channel has an index of 1.

Special values of `allindices` or `alllabels` may be given for the channel <*indexorlabel*>, causing the requested parameters to be reported for all channels. The output for each channel is terminated by the double vertical bar `||`, except for the last channel which is terminated by a `<cr><lf>` pair.

The following parameters give the basic information available for all channels:

> None of these values can be modified by OEM customers.

- `type` is a short, pre-defined name for the installed channel, for example, `temp12` for temperature, `pres25` for pressure, or `irr_03` for irradiance. See Channel types for the full list.
- `module` is the internal address to which this channel responds; it is normally of no interest to OEM customers.
- `status [= <status>]` is another basic parameter which applies to all channels. It allows any individual channel to be turned off or on for the duration of a deployment, and can be modified by OEM customers.
  - `on:` the channel is activated for sampling; its data will appear in streamed output if streaming is enabled.
  - `off:` the channel is not sampled, no data will be streamed for this channel.
- `settlingtime` and `latency` is the minimum power-on settling delay in milliseconds required by this channel, taking into account both the sensor and the interface electronics.
- `readtime` is the typical data acquisition time in milliseconds required by this channel, again taking into account both the sensor and the interface electronics.
  Most channels have a fixed, pre-determined `readtime`, but for some it may be variable. An example would be a channel which supports, and is configured to use, the auto-ranging feature: the `readtime` is longer when the channel is in auto-ranging mode than when operated in fixed-gain mode. The instrument adjusts the reported `readtime` value to reflect the operating mode and status of the channel.
- `equation` is the type of formula used to convert raw data readings to physical measurement units. The values for the core equations are shown below.
  - `tmp,` temperature
  - `lin,` linear
  - `qad,` quadratic polynomial
  - `cub,` cubic polynomial
- `userunits` is a short text string giving the units in which processed data is normally reported from the instrument; for example `C` for Celsius, `V` for Volts, `dbar` for decibars, etc.

- `derived` is a flag which is either `on` or `off` to indicate whether the channel is a derived channel ( `on` ) or a measured channel ( `off` ). This is an intrinsic property of the channel `type` , and cannot be modified; it is for information only. **Not supported by**: RBR*coda* T.ODO.
- `gain` reports the gain setting currently in use by the channel referenced by `<indexorlabel>` . In addition to one of the fixed values from the list reported by the `availablegains` option, the response may indicate `auto` for auto-ranging. In this mode the channel will select the most appropriate gain setting depending on the value of the parameter being measured. If the channel does not support multiple gain settings, the response is `none` . **Not supported by**: RBR*coda* T.ODO.

  The `gain` option may also be used to set the gain used. For a fixed gain setting, the value supplied must be from the list reported by the `availablegains` option. For auto-ranging, use the word `auto` . Although they are typically whole numbers, gains are reported in a floating point format, and may be specified as such, as long as the value appears in the list of available gains.
- `availablegains` reports the gain settings supported by the sensor at channel `<indexorlabel>.` The settings are given as a list of numerical values in order of increasing gain, with a vertical bar `|` separating the values. If the channel at `<indexorlabel>` does not support multiple gain settings, the response is **none**. **Not supported by**: RBR*coda* T.ODO.

> The `availablegains` and `gain` parameters are only available for channel types which support sensors having variable gain, or multiple ranges. This only includes the Seapoint turbidity and RBR*coda* Tu sensors.

- `label` is a short text string without white spaces, comma, equal symbol, or special character, describing the physical parameter measured (example: `temperature_00` ). It is reported when the channel requested is referred by its index.**Not supported by**: RBR*coda* T.ODO.
- `index` is the index of the channel. It is reported when the channel requested is referred by its label. **Not supported by**: RBR*coda* T.ODO.

**Examples**

```
>> channel 1 type
<< channel 1 type = temp15, module = 7, status = on, latency = 250, readtime = 300,
equation = tmp, userunits = C
```

RBR*coda* T.ODO, only the supported parameters are reported.

```
>> channel 1
<< channel 1 type = temp09, module = 1, status = on, settlingtime = 140, readtime = 150,
equation = tmp, userunits = C, derived = off, label = temperature_00
```

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

```
>> channel allindices type
<< channel 1 type = temp12 || 2 type = pres26
```

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

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

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

**Errors**

`Error E0107 expected argument missing`

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

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

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

`Error E0501 item is not configured`

There is a problem with the configuration of the specified channel. Contact RBR for help.

`Error E0505 no channels configured`

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

## 8.7 sensor

**Usage**

`>> sensor <index> [ wavelength ]`

For RBR*coda* T.ODO:

`>> sensor <index> [ foilbatch ]`

**Description**

Returns information about radiometer wavelengths (RBR*coda* rad, RBR*quadrante*). The sensor on the RBR*coda* rad, RBR*quadrante* is identified by an index (an integer). The first sensor has an index of 1.

Channels can be identified either by an index (an integer) or by their label (a meaningful string, for example temperature_00). The first sensor has an index of 1.

The output for each sensor is the wavelength of the channel and its full width at half-maximum (FWHM).

**Examples**

```
>> sensor 1
<< sensor 1 wavelength = 445/10nm
```

RBR*coda* rad

```
>> channel 1 << channel 1 type = irr_07, module = 128, status = on, settlingtime = 50,
readtime = 30, equation = corr_irr2, userunits = uW/cm^2/nm, derived = off, label =
irradiance_00

>> sensor 1 << sensor 1 wavelength = 445/10nm

>> channel 2 << channel 2 type = irr_07, module = 136, status = on, settlingtime = 50,
readtime = 30, equation = corr_irr2, userunits = uW/cm^2/nm, derived = off, label =
irradiance_01

>> sensor 2 << sensor 2 wavelength = 488/10nm

>> channel 3 << channel 3 type = irr_07, module = 152, status = on, settlingtime = 50,
readtime = 30, equation = corr_irr2, userunits = uW/cm^2/nm, derived = off, label =
irradiance_02

>> sensor 3 << sensor 3 wavelength = 560/10nm
```

RBR*quadrante* with three radiometers

**Errors**

`Error E0107 expected argument missing`

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

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

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

`Error E0501 item is not configured`

There is a problem with the configuration of the specified sensor. Contact RBR for help.

## 8.8 channels

**Usage**

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

For the RBR*coda* T.ODO:

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

### Description

Returns general channel information for the sensor. It is a read-only command.

### Parameters

- `count` is the number of channels installed and configured in the sensor.
- `on` is the number of active channels, which excludes any channels turned `off` by the user. See the `status` argument to the channel command.
- `settlingtime` and `latency` is the power-on settling delay in milliseconds; this is the time the instrument will wait after waking from its quiescent state, before attempting to take measurements. It allows all sensors and channel electronics to reach a stable condition. This overall settling time is determined by the longest of all the individual *active* channel delays; channels which are turned **off** do not contribute.
- `readtime` is the overall reading time in milliseconds; this is the additional time the sensor needs to acquire data from all active channels once the settling time has passed. This overall readtime is determined by the longest value of all the individual *active* channels; any which are turned **off** do not contribute.
  Most channels have a fixed, pre-determined readtime, but for some it may be variable. An example would be a channel which supports, and is configured to use, the auto-ranging feature: the readtime is longer when the channel is in auto-ranging mode than when operated in a fixed-gain mode. The instrument adjusts the reported value of the readtime to reflect the operating mode and status of all active channels.
- `minperiod` is the minimum sampling period in milliseconds with the currently active channels. This takes account of the current overall values for the settling and read times, and adds some overhead and safety margin for both fixed and per-channel activities. The value applies to the 1Hz or slower sampling modes supported by all sensors; sensors configured to support fast sampling modes as well may use selected periods less than this value.

### Examples

```
>> channels
<< channels count = 5, on = 5, latency = 250, readtime = 900, minperiod = 1090
```

RBR*coda* T.ODO, only the supported parameters are reported.

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

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

### Errors

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

The command was given with an argument which is unrecognised or misplaced.

`Error E0505 no channels configured`

There is a serious fault with the instrument. Please contact RBR for help.

## 8.9 settings

**Usage**

```
>> settings [ fetchpoweroffdelay | inputtimeout | temperature | pressure |
atmosphere | density | salinity ]
```

For the RBR*coda* T.ODO:

```
>> settings [ fetchpoweroffdelay | salinity | pressure | atmosphere ]
```

**Description**

Reports the values of miscellaneous settings in the sensor.

**Parameters**

- `fetchpoweroffdelay [= <timeoutinmilliseconds>]` is the delay in milliseconds between successful completion of a fetch command, and power to the front-end sensors being removed by the instrument. Power is left on for a short time to avoid excessive power cycling when sending repeated `fetch` commands; this parameter allows that delay to be adjusted. The default value is 8000. This parameter is only available for the RBR*coda T.ODO instruments.*
- `inputtimeout [= <timeoutinmilliseconds>]` specifies the value of a timeout in milliseconds when receiving command input. It is used to temporarily blank other output such as streamed data, and to assist in power saving by turning off the serial communication interface when it is not needed. The default value for all instruments is 10000 (10 seconds). **Not supported by**: RBR*coda* T.ODO.
- `temperature, pressure, atmosphere, density, salinity [= <value>]:` are default parameter values, to be used when the instrument does not have a channel which measures the named parameter, but cross-channel calibration equations require 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 seawater properties:
  - temperature in °C, default value 15.0
  - absolute pressure in dbar, default value 10.132501 (1 standard atmosphere)
  - atmospheric pressure in dbar, default value 10.132501
  - water density in g/cm$^3$, default value 1.026021
  - salinity in PSU, default value 35

**Examples**

```
>> settings
<< settings fetchpoweroffdelay = 8000, salinity = 35.0000, pressure = 10.1325,
atmosphere = 10.1325
```

RBR*coda* T.ODO, only the supported parameters are reported.

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

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

**Errors**

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

The command was given with an argument which is unrecognised or misplaced.

## 8.10 serial

**Usage**

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

For RBRcoda T.ODO:

`>> serial [ baudrate | mode ]`

For the RBRcoda T.ODO the maximum baudrate is `9600` and the only serial mode is `rs232`

**Description**

Reports or sets the parameters which apply to the serial link.
Care must 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 sensor is to recognize it. In particular, changing the `mode` is not recommended unless you have discussed it with RBR first. The individual parameters are described below.

- `baudrate [= <baudrate>]:` corresponds to the baudrate of the serial link. The default baudrate from the factory is 9600.
- `mode [= <mode>]:` allows the electrical interface standard used for the serial link to be changed, available choices being listed below.

Contact RBR for advice before changing the `mode` parameter. Different modes typically require different hardware. An inappropriate change to the `mode` parameter may cripple the sensor communications.

- The most common mode is `rs232`, used on most equipment with serial ports and referred to as RS-232, EIA-232, TIA-232, or variations on one of these depending on the revision. This is the default setting typically shipped from the factory. The implementation of RS-232 on the instrument is always full duplex, with no hardware flow control lines required: **transmit**, **receive**, and **ground** are the three connections needed. If an instrument has been built to use one of the other interfaces, the mode will be correctly set when the instrument is shipped.
- `availablemodes:` reports the list of available modes. **Not supported by**: RBR*coda* T.ODO. The only serial mode allowed for the RBRcoda T.ODO is `rs232`.
- `availablebaudrates:` reports the list of available baudrates. **Not supported by**: RBR*coda* T.ODO. The RBR*coda* T.ODO supports baudrates `9600|4800|2400|1200`, but there is no argument to query this information.

### Examples

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

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

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

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

```
>> serial availablemodes
<< serial availablemodes = rs232
```

### Errors

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

The supplied argument was not a recognised parameter name, baudrate value, or mode setting.

## 8.11 getall

### Usage

`>> getall`

### Description

Reports all the parameters and settings in use.

It is a read-only command, which outputs the result over several lines, unlike standard commands.

The output of this command will vary from one instrument to another. The example below is just an illustration.

### Example

```
>> getall
<< prompt state = off
confirmation state = on
link type = serial
E0102 invalid command 'powerinternal'
memformat type = rawbin00, newtype = rawbin00, availabletypes = rawbin00|none
settings fetchpoweroffdelay = 8000, sensorpoweralwayson = off, temperature = 15.0000,
atmosphere = 10.1325010, pressure = 10.1325, density = 1.0260207, density = 1.0260207,
castdetection = off, inputtimeout = 10000
clock datetime = 20000101010815, offsetfromutc = unknown
```

```
sampling mode = continuous, period = 63, gate = none, userperiodlimit = 31,
availablefastperiods = 500|250|125|63|42|31
deployment starttime = 20000101000000, endtime = 20991231235959, status = streaming
calibration 1 label = temperature_00, datetime = 20000401000000, c0 = 3.5000000e-003, c1
= -250.00002e-006, c2 = 2.7000000e-006, c3 = 23.000000e-009 || calibration 2 label =
pressure_00, datetime = 20000401000000, c0 = 0.0000000e+000, c1 = 1.0000000e+000, c2 =
0.0000000e+000, c3 = 0.0000000e+000, x0 = 0.0000000e+000, x1 = 0.0000000e+000, x2 =
0.0000000e+000, x3 = 0.0000000e+000, x4 = 0.0000000e+000, x5 = 0.0000000e+000, n0 = 3
outputformat type = caltext06, availabletypes = caltext06|caltext08, labelslist =
temperature_00|pressure_00
id model = RBRcoda3, version = 1.000, serial = 092087, fwtype = 105, flavour = rt
info pn = SEN3-M32-F35-SEC33-INT32-ST32-SP31
hwrev pcb = G, cpu = 5528H, sbsl = n/a
channels count = 2, on = 2, settlingtime = 50, readtime = 335, minperiod = 465
channel 1 type = temp12, module = 1, status = on, settlingtime = 50, readtime = 300,
equation = tmp, userunits = C, gain = none, availablegains = none, derived = off, label
= temperature_00 || channel 2 type = pres26, module = 2, status = on, settlingtime = 50,
readtime = 335, equation = corr_pres2, userunits = dbar, gain = none, availablegains =
none, derived = off, label = pressure_00
sensor 1 || sensor 2 serial = H163989
```

**Errors**

`E0102 invalid command <unknown-command-name>`

The response to `getall` may contain error messages about invalid commands or arguments and, in this case, such errors can be safely ignored. They mean that the instrument is configured with certain options disabled.

## 8.12 id

**Usage**

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

**Description**

Identifies the sensor.

It is a read-only command which reports the model name, the version of firmware in the CPU, the unit serial number, firmware type, and may contain the flavour of instrument (rt = realtime). The serial number is always reported using six digits, padded with leading zeroes if necessary.

**Examples**

```
>> id serial
<< id serial = 092087
```

```
>> id
<< id model = RBRcoda3, version = 1.000, serial = 092087, fwtype = 105, flavour = rt
```

**Errors**
None.

## 8.13 info

**Usage**

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

> The RBR*coda* T.ODO does not support the `info` command.

**Description**

Reports the RBR part number of the instrument. This is a read-only command.

**Example**

```
>> info pn
<< info pn = SEN3-M32-F35-SEC33-INT32-ST32-SP31
```

**Errors**
None.

## 8.14 confirmation

**Usage**

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

**Description**

Returns the state of the sensor confirmation responses.

It is normally sent after a parameter has been modified if the state is `on`. If the state is `off`, successful parameter modifications occur without confirmation messages.

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

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

> A request for information or a failed attempt to set a parameter always provokes a response, even when the state is `off`. For example:
> - Requests to simply report a parameter always generate output.
> - Error messages resulting from a *failed* attempt to set a parameter are always sent.
> - Some action commands always generate a confirmation message.
> - The `Ready:` prompt is controlled separately by the prompt command.

Turning confirmation off is not normally recommended, unless there is a very good reason for doing so. For example, if it is interfering with the parsing of responses by an automated system, it may be necessary to suppress it.

**Examples**

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

Confirmation of a parameter change is immediately suppressed.

```
>> confirmation
<< confirmation state = off
>> serial baudrate
<< serial baudrate = 19200
```

While the `confirmation` `state = off`, a request for information always provokes a response.

**Errors**

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

The command was given with an argument which is unrecognised or misplaced.

## 8.15 prompt

**Usage**

`>> prompt [ state ]`

**Description**

Returns the state of the `Ready:` prompt.

It is normally sent by the sensor in response to almost any command after any other output generated by the command is complete.

- `state [= <state>]`
    a. `on`, the prompt is sent
    b. `off`, the prompt is suppressed.

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

**Examples**

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

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

**Errors**

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

## 8.16 config

**Usage**

`>> config restore`

**Description**

Restores factory configuration for the RBR*tridente*. This is a protected command.

> ⚠ This command requires permission. Please contact RBR before attempting to use it.

The instruments may ship with different sets of default coefficients. Use `restore` to recover this information.

- `restore` returns the sensor to factory configuration (calibration coefficients will be untouched).
  The sensor configuration is stored as a binary table in non-volatile memory. When a restore request is received (`config restore`), the backup configuration is read from memory and used to reconfigure the sensor as follows:
  - All sections are copied one-to-one (except for the ID)
  - The channel status (on / off) is reverted back to default
  - The channel gain (for relevant channels) is reverted back to default (automatic for the RBR*tridente*)

  The ID section is not updated. The channel calibration coefficients are not reverted.

**Examples**

```
>> permit command = config
<< permit command = config
>> config restore
```

Factory defaults will be restored from the backup copy.

**Errors**

`Error E0107 expected argument missing`
An argument expected by the instrument was not given with the command; for example, instead of `config restore` there was only `config`.

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

`Error E0111 command failed`
The factory configuration was not stored before restoring or the factory configuration became corrupted.

# 9 Revision history

| Revision No. | Release Date | Notes |
|---|---|---|
| D | 25 June 2026 | Updated the RBRtridente's backscatter channel unit on Realtime data format page. |
| C | 24-November-2025 | Added some clarifying information regarding the RBR*coda* T.ODO to sampling, outputformat, fetch, calibration, channel, channels, settings, serial, and quickstart pages.<br>Added comment to sampling page regarding period values for the RBR*tridente.*<br>Added RBR*coda* chl-*a* to Channel types, Channel labels, and realtime data format.<br>Updated Outputdata with some clarifiying information. |
| B | 06-November-2024 | Added sampling aggregate mode and statistics channels.<br>Added phycoerithrin and phycocyanin channels. |
| A | 06-March-2024 | Initial release |