RBR*coda*

SENSOR USER GUIDE

rbr-global.com

Table of Contents

1 Ruskin 4

2 Revision history 5

3 Warranty statement 6

4 Introduction 7

5 Installation 8

5.1 Install Ruskin on a PC 8

5.2 Install Ruskin on a Mac 9

5.3 Update Ruskin 10

5.4 Uninstall Ruskin 10

6 Provide your feedback 12

7 Quick start 13

7.1 Deploy a sensor 13

7.2 Connecting to the sensor 16

7.3 Data format 20

8 Configure a sensor 22

8.1 RBRcoda deployment 22

8.2 Saved data 22
8.3 Setup 23
8.4 Wave and tide sensors 24

9 User calibration 28
9.1 N-Point calibration 29
9.2 Oxyguard DO calibration 29

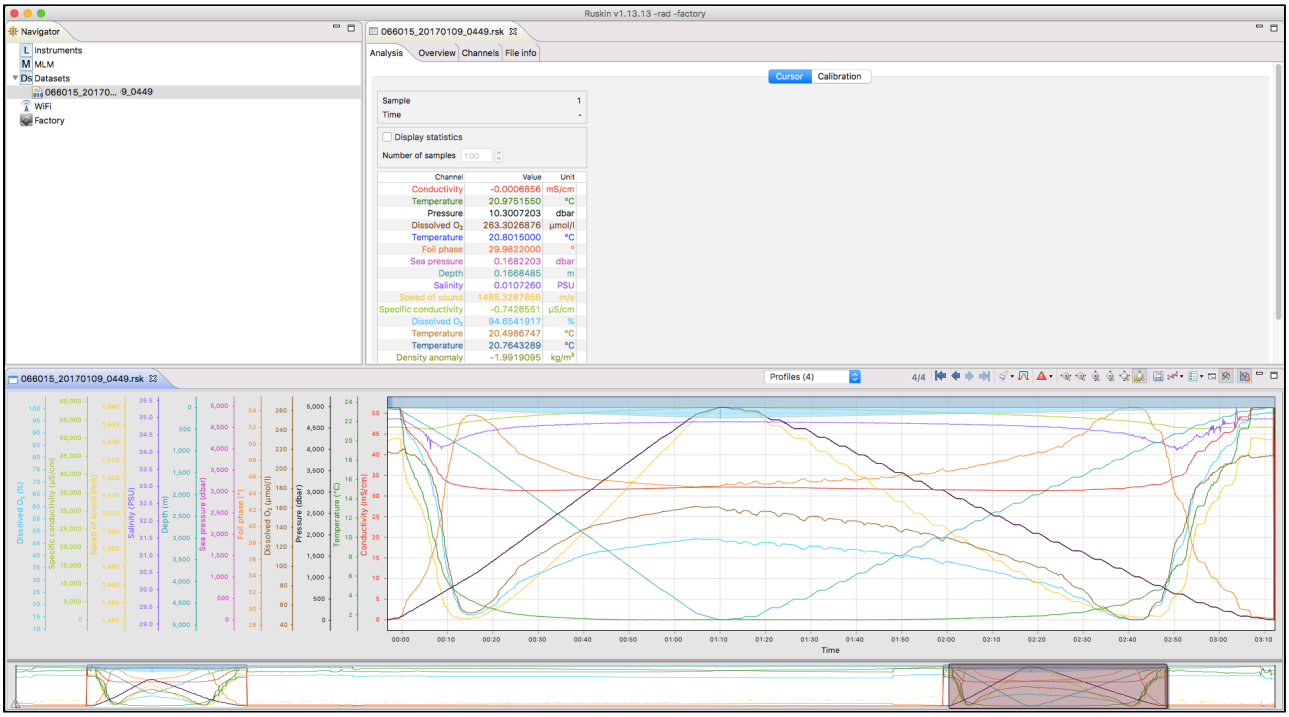
10 OEM Commands reference 32
10.1 Formatting 32
10.2 Supported Channel Types 32
10.3 Commands 33

1 Ruskin

Ruskin is the RBR software that manages your RBR sensors to provide all the data necessary to do your work. Ruskin provides a graphical user interface that makes using the sensors easy. You can use Ruskin to do the following:

- config ure and enabl e multiple sensors
- graphically view data sets
- export data in various formats

Ruskin can be used on PC and Mac.



2 Revision history

Revision No.	Release Date	Notes
1.0	01-Apr-2016	First release
A	09-Mar-2017	Draft Review
B	12-Sept-2017	Final revision for RBRcoda



3 Warranty statement

All sensors manufactured by RBR Ltd. are warranted against defects in workmanship or original parts and materials for one year. Third party sensors (not manufactured by RBR) are limited to the warranty provided by the original manufacturer.

Units suffering from such defects will be repaired or replaced at the discretion of RBR Ltd., provided that the problem has appeared during normal use of the instrument for the purpose intended by us. The liability of RBR Ltd. extends only to the replacement cost of the instrument. The customer will bear all costs of shipment to us for repair; all other costs, including return shipment, will be borne by RBR Ltd.

This warranty does not cover consumables or normal wear and tear, nor does it cover damage caused by negligent use or mishandling. Attempted modification or repair of any unit without the prior consent of RBR Ltd. will immediately void any warranty in force.

Users are expected to maintain a regular program of calibration.

We reserve the right to grant or refuse warranty repairs at our discretion if we consider that there are reasonable grounds for doing so.

4 Introduction

This document introduces you to Ruskin and helps you to use it effectively from the start. It is specifically written for our real-time sensors RBR*coda*.

You can access the Ruskin User Guide on the USB data stick provided when you purchase a sensor, from the Help menu in Ruskin, and on the RBR website, at www.rbr-global.com.

Release notes are automatically displayed each time you install an updated version of Ruskin. The most recent release notes are also available from the Help menu in Ruskin.

5 Installation

5.1 Install Ruskin on a PC

You can install Ruskin on a PC that runs the Windows 7, 8, 8.1, or 10 operating system.

The minimum requirements for Ruskin are:

- OS = Windows 7
- Processor speed = 1.4GHz
- RAM required = 1GB
- Display resolution = 1024x768 recommended
- HDD space for installation = 300MB

Steps

1. Connect the data stick included with your instrument to a USB port.
2. Navigate to the folder Ruskin Installation and double click on the file `RuskinSetup.exe`.
3. Follow the installation wizard. By default, Ruskin will be installed to `C:\RBRRuskin`.
4. The logger uses a USB interface to communicate with Ruskin.
   At the end of the installation, a prompt will appear asking, "Would you like to install the logger driver at this time?"

   > ⚠ Realtime sensors will not require USB drivers to operate, they are only required for our loggers.

5. Click **Yes** to install the drivers.

   > ⚠ You may need to run the setup application as an administrator to install the driver correctly.

A shortcut to Ruskin appears on the desktop and in a **Start** menu folder called RBRRuskin.

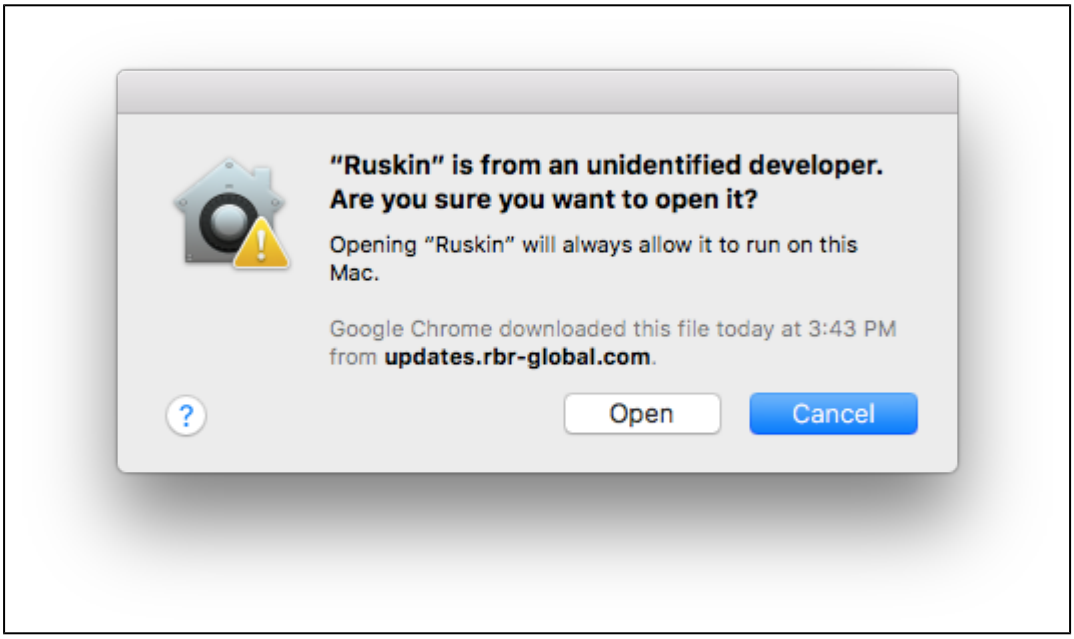
⚠ Please note that the most recent version of Ruskin can be found at https://rbr-global.com/products/software

5.2 Install Ruskin on a Mac

You can install Ruskin on a Mac running OS X 10.5 (Leopard) or later.

Steps

1. Insert the USB data stick in the appropriate USB port.
2. Navigate to the folder OSX and double click on the file `Ruskin.dmg`.
3. When the disk image window opens, drag the Ruskin icon into the applications directory and wait for the copy to complete.
4. To open Ruskin for the first time navigate to your applications directory, locate Ruskin, right click on the icon, and select Open.
5. The dialogue box shown below will prompt you to authorize the opening of Ruskin.



It may be required that you navigate to **System Preferences > Security & privacy** to allow apps downloaded from “**Anywhere**” to complete the installation.



Although you can specify a different folder for the working directory for the software, we recommend that you use the default **Applications** folder.

A folder named RBRRuskin appears in the Applications folder.

You may want to drag the Ruskin.app application to the Dock.

5.3 Update Ruskin

To take advantage of new features and bug fixes, ensure that you are using the most recent version of Ruskin.

It is not necessary to uninstall an older version of Ruskin before installing a newer version. The installation program deletes the older files before installing the newer ones. It does not delete any Ruskin data files or log files.

The most recent version of Ruskin is always available on the RBR website (www.rbr-global.com). However, if you already have an older version of Ruskin installed, Ruskin automatically notifies you that a newer version is available when you start Ruskin. You can check to see if a new version is available from within Ruskin navigating to the menu **Help > Check for updates**. If you have a broadband connection, we recommend that you follow the installation instructions that appear on your computer. Otherwise, request a USB stick from RBR.

If you do not have a broadband connection and/or are unable to install the Ruskin updates, update notifications are available via email. To receive these notifications, send an email to: support@rbr-global.com subject: "Ruskin update request".

5.4 Uninstall Ruskin

If you no longer need to manage RBR instruments from your computer, you can uninstall Ruskin.

⚠ Removing Ruskin will not delete your data files or your diagnostic logs.

It is not necessary to uninstall an older version of Ruskin before installing a newer version. The installation program deletes the older files before installing the newer ones. For more information, see Update Ruskin (page 10).

Windows 7, 8, 8.1, or 10

Go to **Start** > **Control Panel** > **Programs**, and under **Programs and Features**, click **Uninstall a program**. In the list, locate **Ruskin** – click **Ruskin** to highlight it, and then click **Uninstall**.

OS X 10.5 or later

Move the RBRRuskin folder from **Applications** to the **Trash**.

6 Provide your feedback

You can get in touch with RBR in several different ways:

- Send us an email. For a technical question, write to support@rbr-global.com. For general inquiries, use info@rbr-global.com.
- Send us a bug report from within Ruskin itself. Use the **Help** menu > **Comment on Ruskin**. This allows you to include the diagnostic logs, and any other files (RSK datasets, screenshots) that will help us reproduce the problem and help you as quickly as possible.

Steps

1. From the **Help** menu, click **Comment on Ruskin**.
   The Feedback to RBR dialog box appears.
2. Enter your identification information, for example, email address and name, and then summarize your comments.
3. Provide a detailed description and add any attachments, if required.
4. Click **Submit** to submit the report.

7 Quick start

7.1 Deploy a sensor

The subset of commands described below are the parameters required to operate the sensor in streaming or polled modes.

7.1.1 Streaming

The only two parameters needed to operate your sensor in steaming mode, the streaming speed, and the serial baud rate. The sensor will begin streaming data as soon as power is provided.

- The factory streaming speed is the fastest that the sensor is rated for, e.g. |fast 16 will be set to 16Hz.
- The factory baud rate is 9600.

You can communicate with your RBR sensor using Ruskin or a serial terminal.

Steps for communicating using Ruskin

- Connect the sensor to your computer using a serial interface or a serial to USB adapter.
- If you are using Ruskin the sensor will be shown in the **Navigator** window under **Instruments**
    - Click on the sensor that you want to use.
    - In the **Sensor Configuration** window, under **Setup** you will see the details of the sensor, the sampling schedule, the baud rate, and the realtime RSK file location.
    - In the **Plot** window located at the bottom of the Ruskin window you will see the realtime data
- You can change the parameters using the controls on screen and they take effect immediately.
- All settings persist over power cycles.

Steps for communicating using a serial terminal

- **sampling** - Allows various parameters to be reported or set.
  - **period** [=<*period*>] reports or sets the time between measurements. The <*period*> is specified in milliseconds. Values for 1Hz sampling or slower are supported by all sensors, and must be given in multiples of 1000. If the sensor is configured to support fast (sub-second) measurement speeds, the <*period*> must be one of: **500** (2Hz), **250**(4Hz), **125**(8Hz), and **63**(16Hz).
    Examples:

```
>> sampling
<< sampling schedule = 1, mode = continuous, period = 125, burstlength = 60, burstinterval = 300000
```

    The sensor has been programmed for continuous 8Hz sampling. The programmed values of the burst parameters are reported but do not apply to continuous sampling.

```
>> sampling period = 5000
<< sampling mode = 5000
```

    The sensor has been programmed to sample once every 5 seconds

- **serial -** Report or set the parameters which apply to the serial link.
  - **baudrate** [=<*rate*>] the following rates are supported: 9600, 4800, 2400, and 1200. The default shipped from the factory is 9600.
    Examples:

```
>> serial
<< serial baudrate = 9600
```

The sensor has been has been programmed with a baud rate of 9600

```
>> serial baudrate = 4800
<< serial baudrate = 4800
```

The sensor has been programmed to utilize the baud rate of 4800

7.1.2 Polled

To operate the sensor in polled mode, you will first need to disable streaming, then use the fetch command to retrieve a sample from the sensor.

- **stream** - This command can turn data streaming on or off. If the command is given with no parameters, the value of the **state** parameter is reported.
  - **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 is not sent.

Example:

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

- **fetch** - requests an 'on-demand' sample set from the sensor.

Example:

```
>> fetch
<< 1000, 18.1745, 12.7052
```

Additional commands can be found in the OEM section of this guide.

7.2 Connecting to the sensor

Details

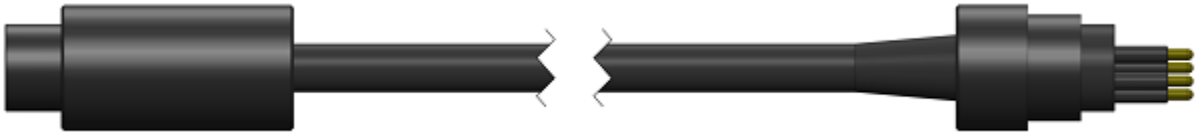
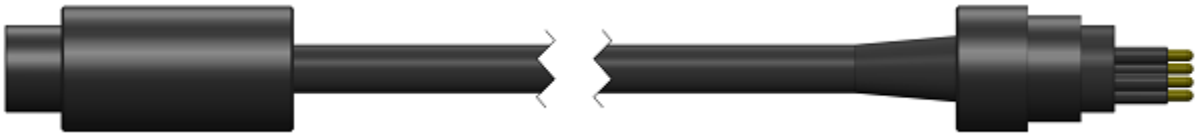
External Power	Requires 6-18V ~4mA
Communication	RS-232 or full duplex RS-485
Data	Polled or autonomous streaming
Baud Rate	1200 to 115k
Sensor connector	MCBH-6MP

7.2.1 Cables

Patch cables are intended for connections between an instrument and a computer. Underwater extension cables may be used for serial output instrument to increase the distance between computer and instrument. RS-232 can be used up to 50m (longer with lower baud rates). RS-485 is the long-distance choice.

Interconnect cables may be used for RS-232 and power, or general analogue signals, typically between two underwater devices.

Underwater Cables

Name	Notes
RS-232 MCIL-6FS to MCIL-6MP, underwater extension cable	Extension cable with RS-232 and power wiring.
	
RS-485FD MCIL-6FS to MCIL-6MP, underwater cable	Extension cable with full duplex RS-485 and power wiring.
	

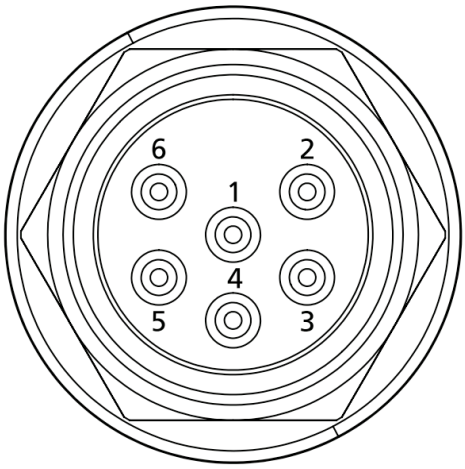
7.2.2 Patch Cables

P/N	Name	Notes
0003664	RS-232 MCIL-6FS to USB Type A , patch cable, 2m	For instruments with RS-232 output (embedded converter). Includes power terminal block.

P/N	Name	Notes
		
0003663	RS-485FD MCIL-6FS to USB Type A , patch cable, 2m	For instruments with RS-485 output (embedded converter). Includes power terminal block.
		
0003970	RS-232 MCIL-6FS to DB9-F, patch cable, 2m	For instruments with RS-232 output. Includes power terminal block.
		
0004126	RS-485FD MCIL-6FS to DB9-F, patch cable, 2m	For instruments with full duplex RS-485 output. Includes power terminal block.
		

7.2.3 Pinout Diagrams

The pinout diagram of the MCBH 6MP connector on the RBR*coda* is shown below.

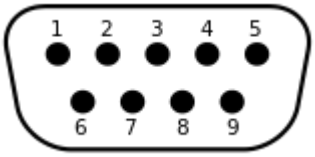


Pin #	RS-232 MCBH Pin Out	RS-485 FD MCBH Pin Out
1	GROUND	GROUND
2	POWER (Nominal 12V)	POWER (Nominal 12V)
3	Tx (Serial data from sensor)	Tx-
4	Rx (Serial data to sensor)	Rx+
5	N/C	Rx-
6	N/C	Tx+

Once power is applied, and streaming is enabled, the sensor will stream the data.

Depending on the ordered configuration, the instrument may be shipped with a RS-232 or RS-485 cable which may have a DB-9 connector (RS-232 P/N 0003970 or RS-485 P/N 0004126) for connecting to your computer via an appropriate adapter, or an embedded converter and USB connector (RS-232 P/N 0003664 or RS-485 P/N 0003663).

The pinout diagram of the DB-9 connector on the MCIL to DB-9 cables supplied by RBR for RS-232 (P/N 0003970) and RS-485 (P/N 0004126) communication is shown below.



Pin #	RS-232 DB-9 Pin Out	RS-485 DB-9 Pin Out
1	N/C	Rx-
2	Tx	Rx+
3	Rx	Tx+
4	N/C	Tx-
5	GROUND	GROUND

7.3 Data format

Sensors will begin streaming data as soon as power is provided.

Below is an example string of data from each variety of RBR*coda:*

Instrument	Example data	Channel list
RBR*coda* ODO	`29000, 23.2868, 200.4000, 93.0000, 245.0000, 29.6900`	Timestamp (ms), Temperature (° C), concentration compensated for salinity (mol/L), saturation (%), concentration uncompensated for salinity (mol /L), phase (°)

Instrument	Example data	Channel list
RBR*coda* T	`29000, 23.2868`	Timestamp (ms), Temperature (°C)
RBR*coda* D	`29000, 10.2484`	Timestamp (ms), Pressure (dbar)
RBR*coda* DO	`29000, 98.8754`	Timestamp (ms), Saturation (%)
RBR*coda* T.D	`29000, 23.2868, 10.2484`	Timestamp (ms), Temperature (°C), Pressure (dbar)

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 channellist**.

Example:

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

8 Configure a sensor

8.1 RBRcoda deployment

There are three precautions you should take to avoid damaging the sensor:

1. Pay attention to the maximum pressure rating. All sensors are individually rated to a maximum depth in meters, this is indicated by the label which is placed on the housing.
2. Avoid physical stress to the sensor. Any type of clamp or bracket which concentrates the stress on the housing is not recommended for use in mooring, mounting, and/or other deployment. Stress due to improper mounting may cause the sensor to leak, resulting in the loss of valuable data or permanent damage to the electronics. RBR can provide proper mooring and mounting clamps suited to your specific application.
3. The sensor is sealed and cannot be opened like our loggers. Any attempt to do so will damage the sensor and it will need to be sent back to RBR for repairs.

8.2 Saved data

8.2.1 Location

When using Ruskin with your sensor, the data is automatically saved to a directory specified for real-time RSK files. To change this location navigate **Options > Preferences > General** and input the desired directory or click "Browse..." to specify the location.

8.2.2 File naming convention

In Ruskin, by default, the name of a data file is composed of the following information:

- The first six digits represent the sensor serial number.
- The next eight digits represent the current year, month, and day.
- The next four digits represent the current time to the minute.
- realtime indicates that the data was captured from a streaming instrument.
- The file extension indicates the file format and should not be changed. If you change it, the file extension that you specify becomes part of the name, and the required extension is appended.

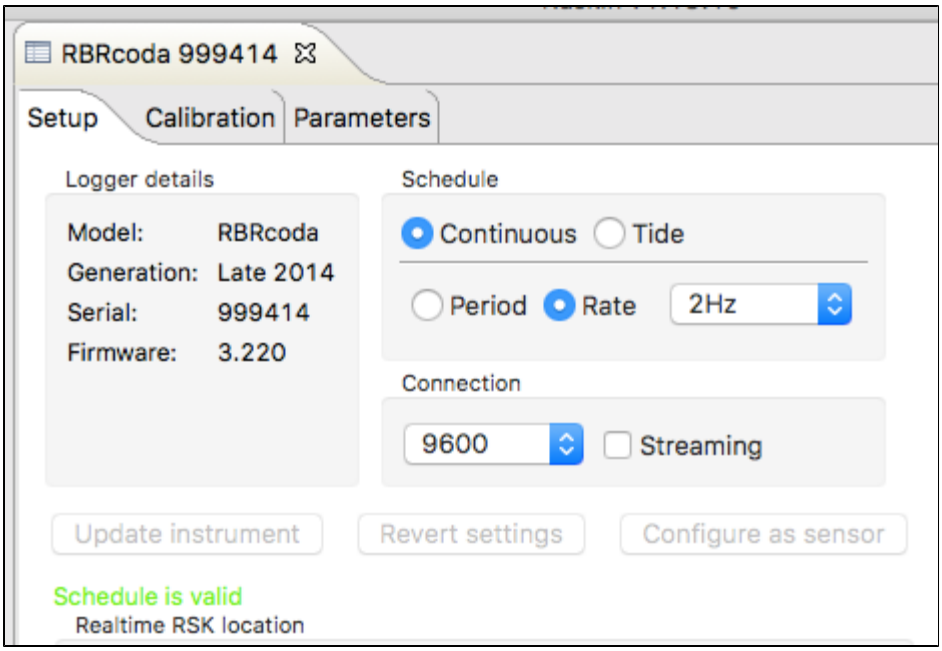
For example, the file named `911936_20090522_1613_realtime.rsk` contains data for a sensor with a serial number of 911936 whose data began streaming in 2009 on May 22 at 4:13 pm.

8.3 Setup

Under the **Setup** tab specify the interval between samples using either the **Period** or **Rate** option. The Period option allows you to set the sampling interval in units of seconds. The Rate option allows you to select between 1 Hz or 2 Hz frequencies.

> ⚠ All |fast16 and |wave sensors have the ability to sample faster than 2Hz. |fast16 and |wave sensors can sample at rates of 4,8, or 16Hz.

The connection baud rate options are 9600, 4800, 2400, and 1200.



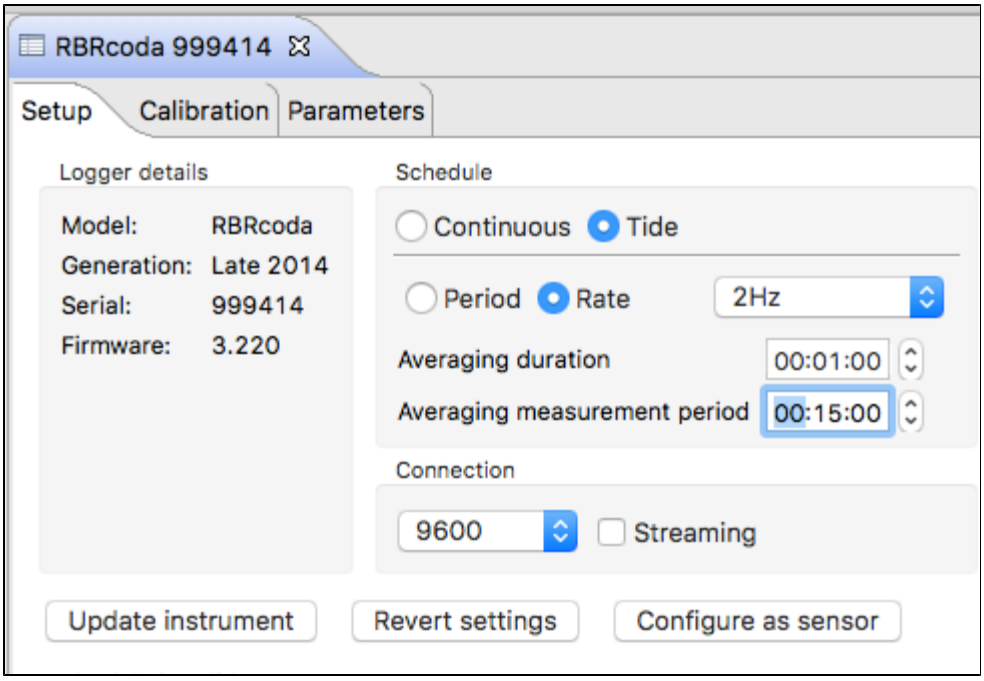
8.4 Wave and tide sensors

8.4.1 Tides

⚠ This section applies to both |tide and |wave variants.

|tide sensors average a burst of pressure readings and stream *only* the resulting average. |wave sensors measure a burst of pressure readings and stream *all* individual samples for post-processing.

From the **Setup** tab select the radio button "Tide" to enable the tide measuring regime.



- Specify how fast you want pressure readings to be taken. Select **Period** for 1s or slower and **Rate** for sub-second sampling rates.
- Specify the **Tidal averaging** duration (how long to average for) and the **Tidal measurement period** (the period on which the average burst should repeat).

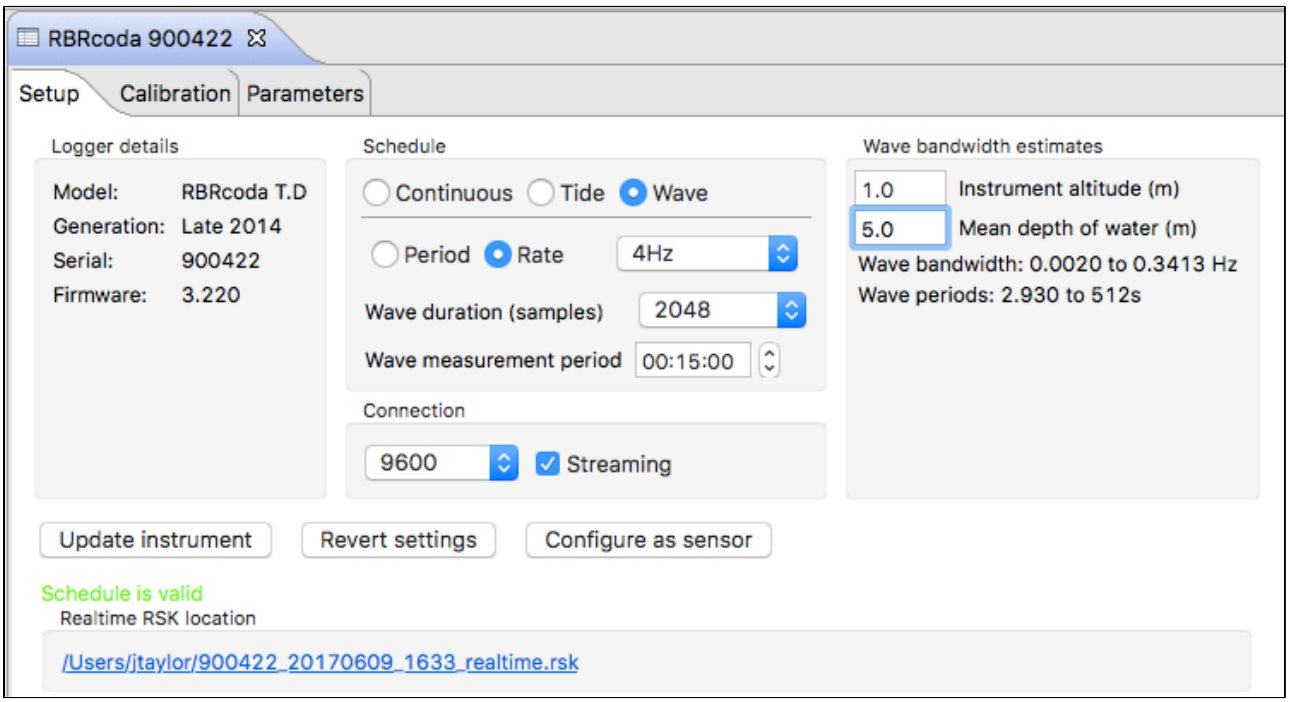
In the above example, the averaging rate is set to 2Hz, the averaging duration is 1 minute, and this sequence is repeated every 15 minutes.

8.4.2 Waves

> ⚠ This section applies *only* to the |wave variants.

A |wave sensor records both wave and tide information, and the configuration is different from other sensors.

From the **Setup** tab select the radio button "Wave" to enable the wave measuring regime.



- In **Sampling regime**, specify how fast you want pressure readings to be taken. Select **Period** for 1s or slower and **Rate** for sub-second sampling rates.
- In **Wave duration (samples)**, select the number of samples that you want to take during a wave burst. The range is between 512 and 32768 samples.
- In **Wave measurement period**, enter the interval between the bursts.
- In **Instrument altitude (m)**, enter the number of meters above the sea or river bed where the sensor will be secured in place. This value is used when making wave bandwidth estimates.

- In **Mean depth of water (m)**, enter the total expected depth of the water where the sensor will be deployed. This value is used to estimate the range of wave frequencies and periods that can be resolved. Ruskin calculates the actual depth from the measured pressure data post-deployment.

The |wave sensor should be fixed to a suitable support below the surface of the water, such as a dock or other rigid mooring. The sensor must not be able to move in the water. The figure below offers a view of the sensor fixed to a dock with a definition of the different water heights.

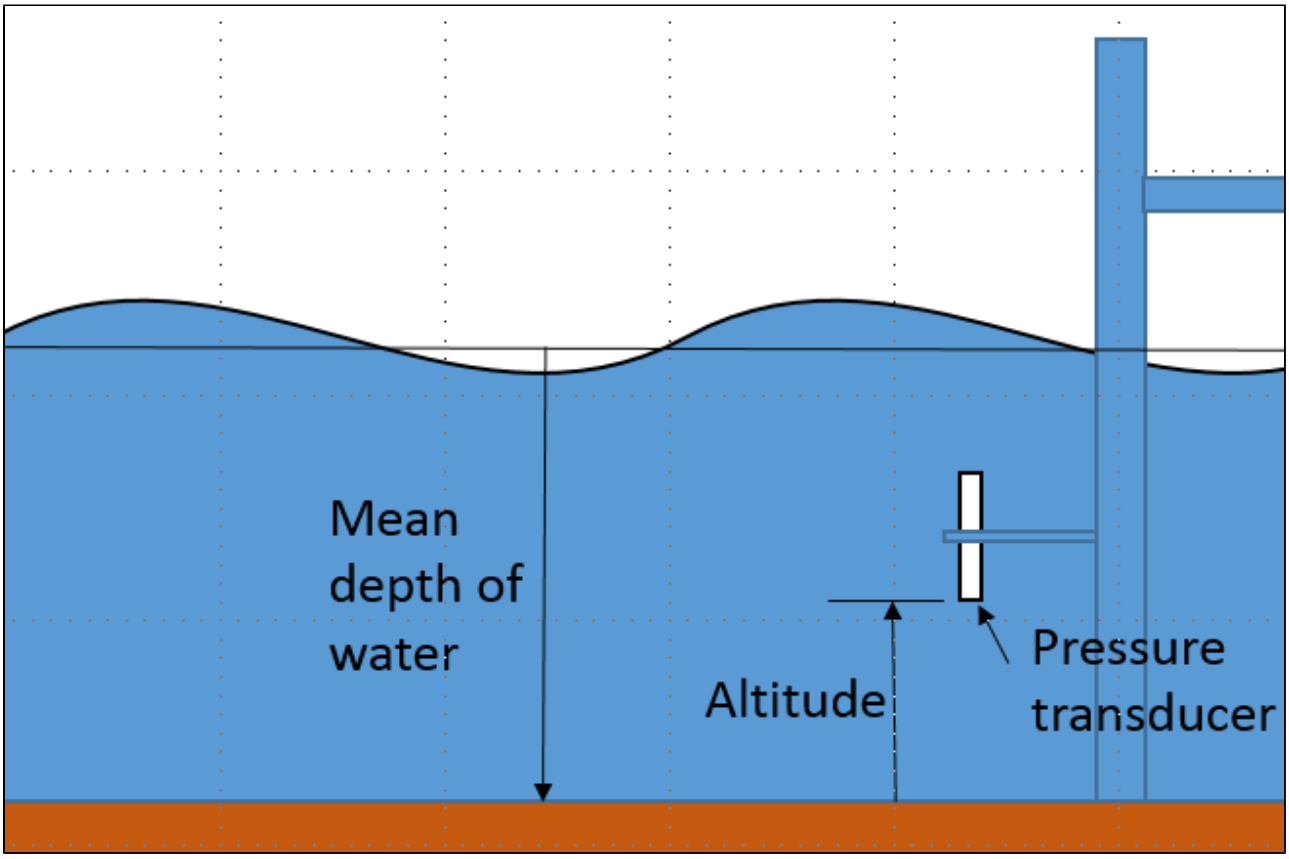


Figure 1. Sensor positioning

For deployment planning, refer to Figure 1. Using the expected mean depth of water (in metres) and the expected altitude (height) of the sensor above the seabed, Ruskin can provide an estimate of the frequencies and periods of the wave that the sensor is able to measure. The sensor measures water depth/pressure by means of a pressure transducer. The physics of what a pressure transducer can 'see' at depth depends on the height of water above the transducer *as well as* the amount of water below the transducer. High frequencies attenuate very quickly with

depth. Figure 2 shows the attenuation with depth as a function of wave period in seconds (*period = 1/frequency*). This graph demonstrates that the placement of the sensor is critical in determining frequencies/periods of the wave data to be captured by the sensor. The pressure transducer may be placed in any orientation.

The basic rule is to place the sensor as close to the surface of the water without the possibility that the sensor will emerge from the water either because of large waves or low tides.

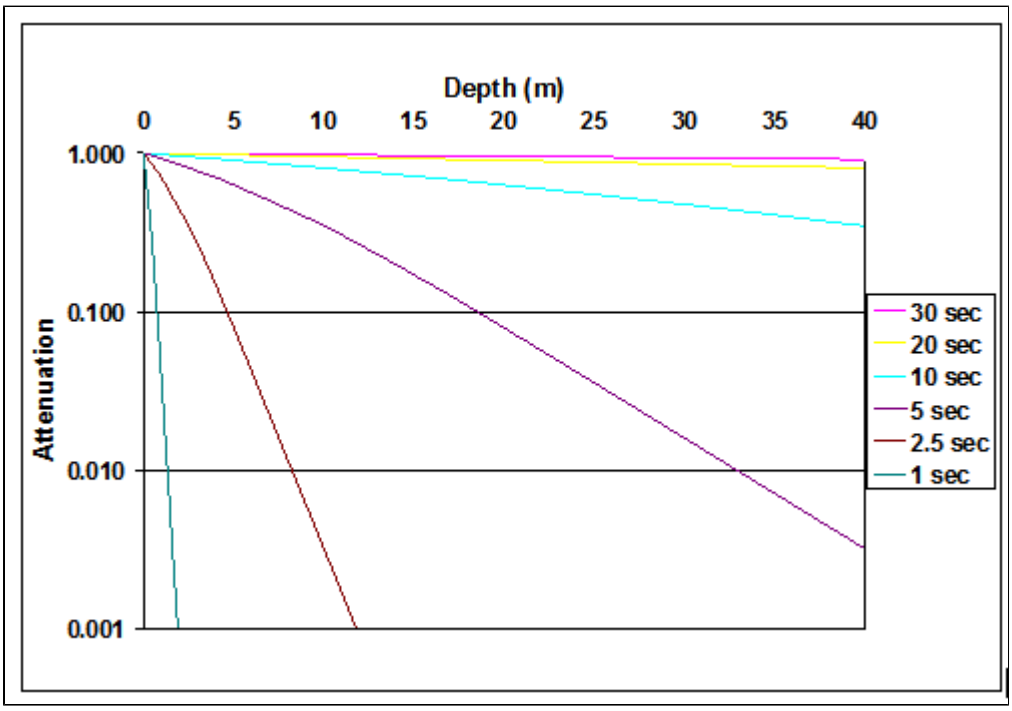


Figure 2. Wave attenuation as a function of depth for various wave periods.

9 User calibration

Calibration coefficients are calculated for each sensor, and the coefficients are stored in the instrument. Calibration certificates are provided for each sensor, and contain both the calibration equation and the coefficients. Hard copies are provided with each shipment and the documents are contained inside the shipping box. Please refer to the calibration certificates for the coefficients and residuals. RBR can replace lost or misplaced calibration certificates.

Change calibration coefficients

Dissolved oxygen sensors can be field calibrated and will require you to update the calibration coefficients for this channels periodically. On occasion, you may also need to manually enter new coefficients, although this is not recommended for factory calibrated sensors (for example T or D) unless instructed by RBR.

Steps

You can view static information about an instrument at any time as follows:

1. In the **Navigator** view, under the **Instruments** list, click the appropriate logger.
2. Click **Analysis** tab > **Calibration** tab to show the current calibration coefficients.
3. To manually change a coefficient, click on the appropriate entry in the table (C1, C2, C3, etc.). The current entry will be highlighted, and the new value can be typed.
4. If a two point calibration has been performed, and calculated coefficients have been copied, right click on either the **Time** or **Parameter** entry for the parameter you wish to modify. Select **Paste to selected row** from the drop-down menu.
5. Click **Store calibration** to write the calibration coefficients to the logger.
6. If you need to revert to previous coefficients, click **Revert calibration**.

⚠ If you do not click **Store calibration**, the coefficients will not be written to the instrument, and will be lost once your session is closed.

9.1 N-Point calibration

Sensors such as Dissolved Oxygen (Oxyguard (page 29)) generate a voltage output that is proportional to the value of the parameter being measured. To calibrate these sensors, Ruskin offers an N-point calibration method to generate calibration coefficients.

9.2 Oxyguard DO calibration

> ⚠ The Oxyguard DO sensor has a true zero point, and therefore it can be calibrated using the single-point calibration method using a reading at 100% oxygen concentration only. The 100% calibration should be performed at the expected temperature and salinity of the deployment environment.

Equipment

1. Two Large mouth beakers
2. Sodium sulphite Na_2SO_3
3. Aquarium air pump
4. Magnetic stirrer

Preparing solutions

Reference Point 1 solution – Oxygen saturated solution at expected temperature and salinity of deployment environment.

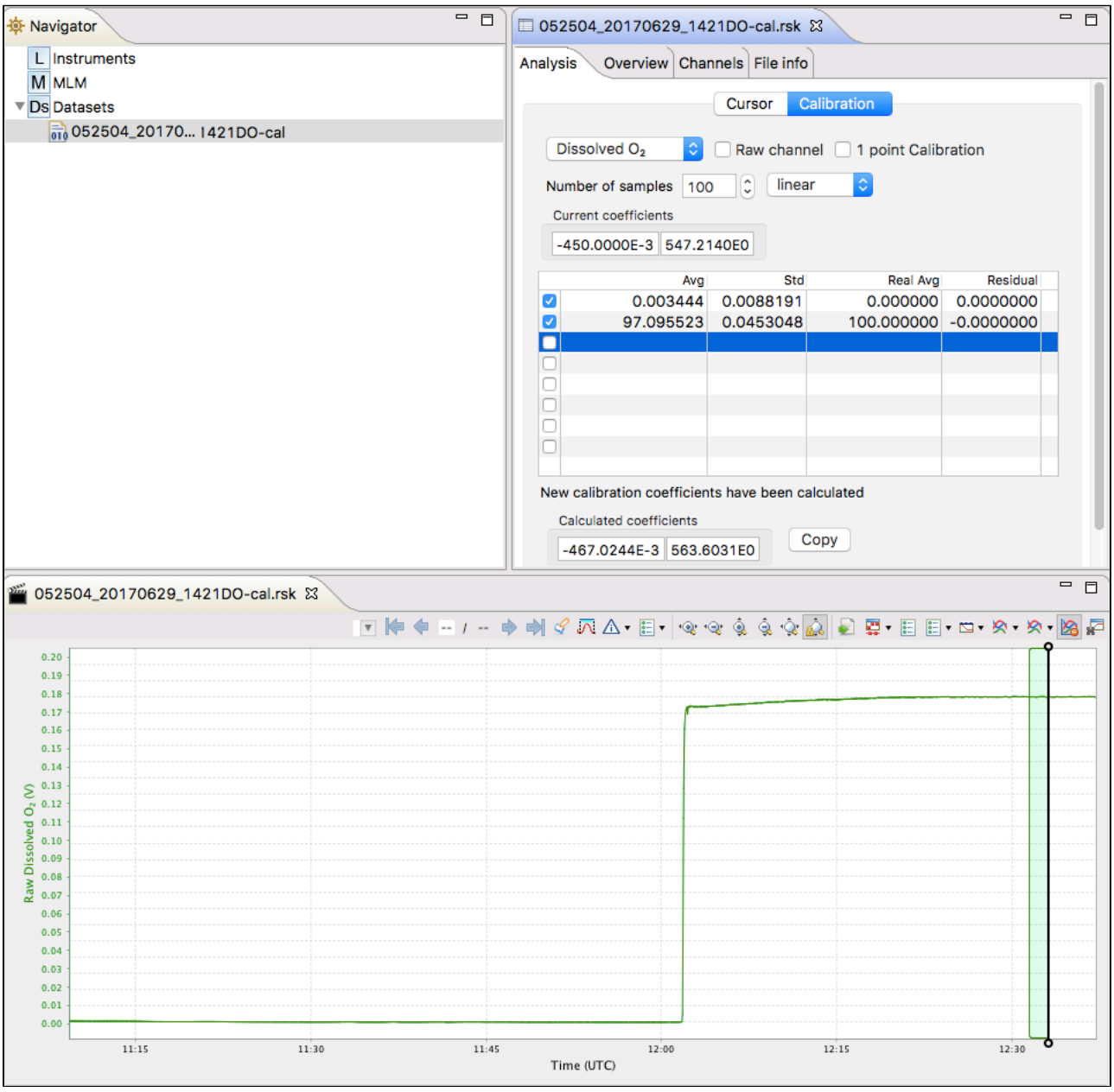
1. Fill the large beaker with 2 L of water. And install magnetic stirrer.
2. Bubble air through the water using an air pump (an aquarium air pump would work).
3. Switch ON the air-pump and the magnetic stirrer.

Reference Point 2 solution – Dissolved oxygen concentration of zero

1. In a beaker, dissolve approximately 5 tsp of sodium sulfite (Na2SO3) into 500 mL tap water.
2. Mix the solution thoroughly with a magnetic mixer. The solution will be oxygen-free after 15 minutes.

Steps

1. In Ruskin, configure the instrument to sample at a fast rate, between 6 Hz and 3 seconds.
2. Submerge the dissolved oxygen sensor in the Reference Point 1 solution for at least 15 minutes near the stirrer so that it is in the best mixed area of the bath.
3. Take sample readings for at least 15 minutes for the 100% calibration point, making note of the time that the sample is being measured.
4. Submerge the dissolved oxygen sensor in the Reference Point 2 solution for at least 15 minutes for the 0% calibration point, making note of the time that the sample is being measured.
5. Retrieve the data from save location described in Download 1 (page 22).
6. The calibration data should now be displayed in the **Plot** view. In the **Properties** view, go to **Analysis** tab > **Calibration** tab.
7. Select the dissolved oxygen sensor type from the drop-down list.
8. The **Number of Samples** spinner box is automatically set to **100**.
   This value is the number of sample points Ruskin will average the calibration coefficients for the sensor. Typically, this value should be in the range of 50 to 100 samples.
9. Click on a stable point in the **Plot** view corresponding to 100% oxygen. In the table in the first row, under **Real Avg**, enter 100 and press enter.
10. Select the check box in the second row in the table, then click on a stable point in the **Plot** view corresponding to 0% oxygen. In the table in the second row, under **Real Avg**, enter 0 and press enter.
11. Ruskin automatically calculates the calibration coefficients, and these values appear in **Calculated coefficients**. Clicking the **Copy** button saves the new calibration coefficients to the clipboard.
12. Follow the steps in Change calibration coefficients (page 28) to update the coefficients for this sensor in the instrument.


Navigator
Instruments
MLM
Datasets
052504_20170... 1421DO-cal
052504_20170629_1421DO-cal.rsk
Analysis
Overview
Channels
File info
Cursor
Calibration
Dissolved O2
Raw channel
1 point Calibration
Number of samples
100
linear
Current coefficients
-450.0000E-3
547.2140E0
Avg
Std
Real Avg
Residual
0.003444
0.0088191
0.000000
0.0000000
97.095523
0.0453048
100.000000
-0.0000000
New calibration coefficients have been calculated
Calculated coefficients
-467.0244E-3
563.6031E0
Copy
052504_20170629_1421DO-cal.rsk
Raw Dissolved O2 (V)
Time (UTC)
11:15
11:30
11:45
12:00
12:15
12:30

10 OEM Commands reference

10.1 Formatting

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

10.2 Supported Channel Types

The following is a list of the channel types supported at the time of writing this document. These type names are used by the channel 2 1 (page 36) command.

Key (implemented)	Units	Manufacturer	Description
doxy12	%	Oxyguard	Dissolved oxygen saturation, RBR*coda* DO
doxy21	M	RBR	Dissolved oxygen concentration compensated, RBR*coda* ODO
doxy22	%	RBR	Dissolved oxygen saturation derived from concentration via Gordon and Garcia, RBR*coda* ODO

Key (implemented)	Units	Manufacturer	Description
doxy24	M	RBR	Dissolved oxygen concentration uncompensated, RBR*coda* ODO
opt_05	°	RBR	Calibrated phase, RBR*coda* ODO
par_02	mol/m²/s	Licor	PAR (photosynthetically active radiation)
pres20	dbar	RBR	Pressure (cubic temperature correction), RBR*coda* D
pres21	dbar	RBR	Pressure (cubic temperature correction), dual channel RBR*coda* T.D
temp02	°C	RBR	Temperature RBR*coda* T
temp12	°C	RBR	Temperature, dual channel RBR*coda* T.D
temp15	°C	RBR	Temperature, RBR*coda* ODO
turb05	NTU	Seapoint	Turbidity, RBR*coda* Tu

10.3 Commands

10.3.1 Configuration Information

10.3.1.1 calibration

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

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

Some sensor types have complicated equations with many coefficients, and the equation may also use the output of one or more of the other channels in the 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 indices. The purpose and function of each group will be described below. The groups may also be referred to by name; **c**, **x** or **n**.

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

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

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

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

Descriptions of the individual parameters are given below.

1. **type** is a read-only parameter created during factory configuration; it can not be modified. It describes the channel type and determines the calibration equation used, and hence the number and type of coefficients required.
2. **datetime** is reported and set using a <*YYYYMMDDhhmmss*> format. It is the date and time of the most recent calibration change for the channel, and is a required parameter when setting any of the calibration coefficients.

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

Equations which use the **x0, x1**… coefficients will require at least one '**n**' index. The sensor 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 sensor which input channels to use.

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

Examples

```
>> calibration 1

<< calibration 1 type = volt04, datetime = 20110518175005, c0 = 9.9876543e+000, c1 = 7.5642301e+000
```

```
>> calibration 1 datetime = 20120503134201, c0 = 9.9873456, c1 = 7.564
<< calibration 1 type = volt04, datetime = 20120503134201, c0 = 9.9873456e+000, c1 = 7.5640000e+000
```

Errors:

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

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

The supplied argument was not valid; examples include:

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

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

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

10.3.1.2 channel

Usage

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

Description

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

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

1. **type** is a short, pre-defined 'generic' name for the installed channel; for example:
    a. **temp08** RBR*coda* temperature,
    b. **pres21** RBR*coda* pressure,
2. **module** is the internal address to which this channel responds; it is normally of no interest to end users.
3. **latency** is the minimum power-on settling delay in milliseconds required by this channel, taking into account both the sensor and the interface electronics.
4. **readtime** is the typical data acquisition time in milliseconds required by this channel, again taking into account both the sensor and the interface electronics.
5. **userunits** is a short text string giving the units in which processed data is normally reported from the logger; for example **C** for Celsius, **dbar** for decibars, etc. Presently this is a factory-set field representing the fundamental units in which the channel is calibrated; support for user-selectable units is planned in the future.
6. **derived** is a flag which is either **on** or **off** to indicate whether the channel is a derived channel (**on**) or a measured channel (**off**). This is an intrinsic property of the channel **type**, and can not be modified: it is for information only.
7. **gainsavailable** reports the gain settings supported by the sensor at channel <*index*>. The settings are given as a list of numerical values in order of increasing gain, with a vertical bar character '**|**' separating the values. If the channel at <*index*> does not support multiple gain settings, the response is **none**.

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

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

    The **gain** option may also be used to set the gain used. For a fixed gain setting, the value

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

> ⚠ The **gainsavailable** and **gain** parameters are presently only available for the Seapoint turbidity sensor.

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

-
  - **on**: the channel is activated for sampling, and its value will appear in reported output if streaming is enabled.
  - **off**: the channel is not sampled, no data will be streamed for this channel.

> ⚠ If the status of a channel is changed the timestamp starts again at 0 and carries on forward.

Examples

```
>> channel 1
<< channel 1 type = temp08, module = 1, status = on, latency = 60, readtime = 320, userunits = C
```

```
>> channel all type
<< channel 1 type = temp08 | 2 type = pres21
```

```
>> channel 1 gainsavailable
<< channel 1 gainsavailable = 1.0|5.0|20.0|100.0
```

```
>> channel 1 gain
<< channel 1 gain = auto
```

```
>> channel 1 gain = 20
<< channel 1 gain = 20.0
```

Errors:

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

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. Sensor channels are numbered from 1 to N; zero is not valid.

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

E0505 no channels configured
There was a problem reading or modifying some configuration data for the specified channel; typically in response to accessing gain control information for those channels which support it. Please contact RBR Ltd for help.

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

10.3.1.3 channels

Usage

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

Description

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

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

Examples

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

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

Errors:

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

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

10.3.1.4 sensor

Usage

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

Description

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

This commands purpose is to manage miscellaneous information relating to the *sensor* associated with a given channel, as opposed to any property of the channel itself; the distinction is rather fine. In general the **sensor** command is used for information which belongs with a particular sensor, but which the instrument does not need to know; a good example would be the sensor's serial number.

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

Examples

```
>> sensor 4
<< sensor 4 foilbatch = 150608-001
```

```
>> sensor 4 foilbatch
<< sensor 4 foilbatch = 150608-001
```

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.

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

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

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

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

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

10.3.1.5 settings

Usage

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

Description

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

1. **fetchpoweroffdelay** is the delay in seconds between successful completion of a fetch command, and power to the front end sensors being removed by the logger. Power is left on for a short time to avoid excessive power cycling when sending repeated fetch commands; this parameter allows that delay to be adjusted. The default value is 10.

2. **temperature**, **pressure**, **conductivity**, **atmosphere**, **density, salinity**: these are default parameter values, to be used when the instrument does not have a channel which measures the named parameter, but one or more cross-channel calibration equations requires it as an input.
   When specifying a value, any simple numeric format compatible with floating point representation may be used; for example 11, 11.000 or 1.10e+1 would all be accepted. The units of these parameter values are implicit, and *must* be as shown below. If these parameter values are never explicitly set, they will have default values based on standard sea water (salinity = 35PSU, temperature = 15°C, hydrostatic pressure = 0 dbar), and one standard atmosphere for atmospheric pressure. The parameters for which setting are available vary from one sensor type to another, here is the complete list.
   a. temperature in °C, default value 15.0
   b. absolute pressure in dbar, default value 10.132501 (1 standard atmosphere)
   c. atmospheric pressure in dbar, default value 10.132501
   d. conductivity in mS/cm, default value 42.914
   e. water density in g/cm3, default value 1.026021
   f. salinity in PSU, default value 35

Examples

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

```
>> settings density = 1.0260209
<< settings density = 1.0260209
```

Error:

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

10.3.2 Communications

10.3.2.1 serial

Usage

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

Description

This command can be used to either report or set the parameters which apply to the serial link. Care must be taken when issuing commands over the Serial link 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. The individual parameters are described below.

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

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 a sensor has been built to use one of the other interfaces, the mode will be correctly set when the sensor is shipped.

- **rs232**: This is the legacy standard used by default on most equipment with serial ports, referred to as RS-232, EIA-232, TIA-232, or variations on one of these depending on the revision, but for most practical purposes they are interchangeable. The sensor's implementation of RS-232 is always full duplex, with no hardware flow control lines required: transmit, receive and ground are the three connections needed.
- **rs485f**: This is the full duplex version of the RS-485 standard (also EIA-485, TIA-485, etc), which permits higher speeds and/or longer distances than RS-232. A five-conductor cable is required; two lines each for both receive and transmit, plus a ground connection. In most cases, a simple cable will work, but at extreme speeds and distances, the transmit and receive line pairs may require impedance matching termination components. The instrument does not include these, as they will be specific to each individual installation.
- **rs485h**: This mode is planned, but not yet supported on any sensors. It is the half-duplex version of RS-485, which requires only three connections: ground plus a data line pair which is used for both receive and transmit.

Examples

```
>> serial
<< serial baudrate = 9600
>> serial baudrate = 4800
<< serial baudrate = 4800
```

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

Errors:

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

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

10.3.3 Data Sample

10.3.3.1 fetch

Usage

>> fetch [sleepafter = true|false]

Description

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

Examples

```
>> fetch
<< 1000, 18.1745, 12.7052
```

Errors:

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

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

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

10.3.4 Other Information

10.3.4.1 altitude

Usage

>> altitude [= <value>]

Description

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

Reports or sets the 'altitude', or height above the sea bed, at which the sensor 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 sensor, and if wave analysis is not required the parameter can be ignored.

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

Examples

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

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

Errors:

E0102 invalid command '<unknown-command-name>'
The sensor is not configured to support the **wave** sampling mode.

E0108 invalid argument to command: '<invalid-argument>'
The argument given was not a valid number.

10.3.4.2 help

Usage

>> help [<command-name>]

Description

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

Examples

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

Error:

E0102 invalid command '<unknown-command-name>'
Help was requested for an unknown command.

10.3.4.3 id

Usage

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

Description

This is a read-only command which identifies the sensor, reporting 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 = 092012
```

```
>> id
<< id model = RBRcoda, version = 3.100, serial = 092012, fwtype = 102, flavour = rt
```

Errors:

Errors

None.

10.3.5 Real Time Data

10.3.5.1 outputformat

Usage

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

Description

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

The parameters currently supported are:

1. **channelslist** This reports a list of names and units for the active channels, in order. This list is helpful in identifying the channel corresponding to each value in the transmitted data. Any channels which have been turned **off** are excluded. The list will expand as support is added for more sensor types, so not all firmware versions will support every sensor type listed. If there is more than one channel of a particular type, the same information is reported for all of them.
   The list of all possible names is given below in alphabetical order. Some are quite generic, others are very sensor specific :
   - depth
   - 02_air_saturation
   - 02_concentration
   - uncompensated_O2_concentration
   - phase
   - PAR
   - period
   - pressure
   - temperature
   - turbidity
   - voltage

Examples

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

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

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. Sensor channels are numbered from 1 to N; zero is not valid.

10.3.5.2 stream

Usage

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

Description

This command can turn 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 baud rate and mode, are accessed using the serial 2 (page 44) 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 is not sent.

Examples

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

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

Error:

E0108 invalid argument to command: '<invalid-argument>'
The supplied argument was not valid or out of place.

10.3.6 Schedule

10.3.6.1 sampling

Usage

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

Description

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

1. **mode** [= continuous | burst | wave | average | tide | regimes] reports or sets the sampling mode for the current schedule.
    a. **continuous** mode is supported by all sensors; measurements are taken and stored at the specified period between the start and end times.
    b. **burst** mode is available for sensors which are so configured in the factory; measurements are taken and stored in bursts between the start and end times. The time between the start of two consecutive bursts is given by the **burstinterval**, the number of measurements in the burst is given by the **burstlength**, and the time between measurements within the burst is given by the **period**.

c. **wave** mode is available for sensors which are so configured in the factory, and with one exception is identical to **burst** mode as far as the sensor's behaviour is concerned. The two modes are distinct to allow host software to process the burst measurements for wave applications. The exception is that the **altitude** command is available *only* when the sensor is capable of the **wave** sampling mode.

d. **average** mode is available for sensors which are so configured in the factory. It works identically to **burst** mode, except that instead of storing every measurement in the burst, the average value of all measurements is computed and stored as a single sample value at the end of the burst.

e. **tide** mode is available for sensors which are so configured in the factory, and is identical to **average** mode as far as the sensor's behaviour is concerned. The two modes are distinct to allow host software to process the data for tide monitoring applications.

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

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

   If the sensor is configured to support fast (sub-second) measurement periods for the selected mode, the <*period*> must correspond to an exact frequency in Hz, to the nearest millisecond. Permitted values may be further constrained as follows:
   |fast 16 - **500**(2Hz), **250**(4Hz), **125**(8Hz), and **63**(16Hz)

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

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

> ⚠ If the mode or period is changed the timestamp starts again at 0 and carries on forward.

3. **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 sensor can be enabled for sampling the value set must also be consistent with the measurement **period** and **burstlength**.

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

   > ⚠ The following read-only parameter is not available on all sensors.

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

Examples

```
>> sampling
<< sampling schedule = 1, mode = continuous, period = 250, burstlength = 60, burstinterval = 300000, gate = none
```

The sensor has been programmed for continuous 4Hz sampling. The programmed values of the burst parameters are reported but do not apply to continuous sampling. No gating condition is in force.

```
>> sampling mode = average
<< sampling mode = average
>> sampling
<< sampling schedule = 1, mode = average, period = 125, burstlength = 60, burstinterval = 300000, gate = none
```

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

```
>> sampling mode = burst, burstinterval = 600000
<< sampling mode = burst, burstinterval = 600000
>> sampling
<< sampling schedule = 1, mode = burst, period = 250, burstlength = 60, burstinterval = 600000, gate = none
```

The mode is changed to burst recording and the burst interval to ten minutes; the sensor is now programmed to take a burst of 60 measurements at 4Hz every ten minutes, storing all measurements in memory.

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

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

Errors:

E0108 invalid argument to command: '<invalid-argument>'
The command was given with an argument which is unrecognized or has an invalid value; for example "sampling mode = mean", or "sampling schedule = 2".

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

10.3.7 Security and Interaction

10.3.7.1 confirmation

Usage

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

Description

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

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

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

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

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

Examples

```
>> confirmation
<< confirmation state = on
>> channel 1 status
```

```
<< channel 1 status = on
>> channel 1 status = off
<< channel 1 status = off
>> channel 1 status = enabled
<< E0108 invalid argument to command: 'enabled'
```

Confirmation of a parameter change is immediately suppressed.

The following example commands are all with confirmation state = off:

```
>> confirmation
<< confirmation state = off
>> channel 1 status
<< channel 1 status = on
>> channel 1 status = off
<<
>> channel 1 status = enabled
<< E0108 invalid argument to command: 'enabled'
```

A request for information always provokes a response.

Response suppressed if the parameter is successfully changed.

Error:

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

10.3.7.2 prompt

Usage

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

Description

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

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

Examples

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

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

Error:

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

10.3.8 Multidrop behaviour

10.3.8.1 Sensor ID auto-discovery procedure

To use the auto-discovery feature on an RS-485 multidrop communication line with multiple sensors attached, the following commands have to be sent out by the main node:

- discovery (page 58)
- quiet (page 59)
- resume (page 59)

1. Send the **>>discovery** command; sensors will reply with their serial numbers.

2. Wait 3 seconds for all sensors to respond.
3. Every sensor that replies must be told to ignore any further >>**discovery** commands. Use the targeted **>>@xxxxxx quiet** command, where **xxxxxx** is the serial number.
4. Repeat steps 1 - 3 until there are no more replies to the **>>discovery** command (all sensors have been identified).
5. The **>>resume** command is optional; it restores the sensor's ability to respond to **>>discovery** commands. It can be used if the auto-discovery process needs to be repeated.

NOTE: Although each sensor will respond to the **>>discovery** command using a different time delay, data collisions can still happen occasionally resulting in random characters due to overlapping data.

10.3.8.2 Targeted commands

Once all sensors are identified, use targeted commands to address each sensor. If a targeted command is not used all sensors will reply in unison resulting in an unintelligible message.

```
>>@999512 fetch
<<4354750, 25.001, 10.213
>> fetch
<<¡rôø˜ªøWö$ö›ö
```

10.3.8.3 discovery

Usage

>> discovery discoverySupported since firmware : 3.210

Description

This is a read-only broadcast command used for sensor identification during the auto-discovery process on a multidrop RS-485 communication line.

Upon receiving the >>**discovery** command, each sensor on the multi-drop will respond at a different time (random response delay to prevent data collisions on the communication line).

The response delay of a sensor can be anywhere from milliseconds to 2.5 seconds. The node that issued the **>>discovery** command (usually the main node) should allow for enough time for all sensors to respond.

```
>> discovery
.
% some random time delay
.
<< serial = 009875
```

Errors:

None

10.3.8.4 quiet

Usage

>> quiet @xxxxxx quietSupported since firmware : 3.210

Description

This is a no-response targeted command, it is used to communicate to a single sensor on a multidrop RS-485 line that it should ignore any further **>>discovery** commands.

Like any targeted commands, the @ symbol followed by a the sensor's serial number should be placed in front of the **quiet** command.

```
>>@090254 quiet
<<
```

Errors:

None

10.3.8.5 resume

Usage

>> resume resumeSupported since firmware : 3.210

Description

This is a no-response command, it is used to return a sensor back to its default setting (reply to **>>discovery** commands). It can be used either as a broadcast or as a targeted command.

```
>>resume
<<
>>@090254 resume
<<
```

Errors:

None