# Wave Glider® SV2 Integrated Sensor User Guide

## Turner C3$^{TM}$ Submersible Fluorometer

**Revision A**
**October 2013**
**Part Number 05573**

# Wave Glider® SV2 Integrated Sensor User Guide: Turner C3™ Submersible Fluorometer



## Trademarks

Liquid Robotics®, LRI, Wave Glider®, Wave Glider Management System, WGMS, and SHARC® are worldwide trademarks of LRI, Incorporated. Other products and company names mentioned herein may be the trademarks of their respective owners.

## Patents

The Wave Glider product is subject to the following patents: Chile (48.628), China (2011 101405179), Egypt (25194), Indonesia (P0027767), Israel (192828), Japan (2009533257), New Zealand (570562 and 592743), Singapore (144487), South Africa (200806769), U.S. (7,371,136, 7,641,524, 8,043,133, and 8,287,323). Additional U.S. and non-U.S. patent applications are pending. All aspects of the Wave Glider product have been developed at private expense by LRI, Inc.

## Warranty

The information contained herein is subject to change without notice. The only warranties for LRI, Inc. products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as a warranty. LRI, Inc. is not liable for technical or editorial errors or omissions contained herein.

## Acknowledgements

Workhorse ADCP Monitor is a trademark of Teledyne RD Instruments, a Teledyne Technologies Company.

## Revision History

| Document Part Number | Revision | Publication Date | Changes |
|---|---|---|---|
| 05573 | A | October 2013 | This is the initial release of this document. |

# Table of Contents

# List of Figures

# List of Tables

# Preface

This guide provides information on operating the Liquid Robotics Wave Glider SV2 with the Turner Designs C3™ Submersible Fluorometer (C3 sensor) integration option. It provides an overview of the mechanical, electronic, and software aspects of the Wave Glider SV2/C3 sensor integration. It includes descriptions of the command structure, data formats, and data retrieval methods supported by the Wave Glider SV2 when operating the C3 sensor. This guide also includes specific considerations for the C3 sensor during operational planning, launch, recovery, storage, troubleshooting, and maintenance.

This guide assumes a working familiarity with the Liquid Robotics Wave Glider SV2 and the C3 sensor. It focuses primarily on information unique to the C3 sensor integration with the Wave Glider SV2. This manual also assumes that the C3 sensor has been calibrated according to the *Turner Designs C3 User's Manual*. Make sure you have the most recent versions of the following documents available when operating the Wave Glider SV2 with an integrated C3 sensor:

- *Turner Designs C3 Data Sheet*
- *Turner Designs C3 User's Manual*
- *Turner Designs C3 Quick Start Guide*
- *Turner Designs Optical Specification Guide*
- *Wave Glider SV2 Interface Control Description* (Liquid Robotics)
- *Wave Glider User Manual* (Liquid Robotics)

**Turner Designs C3 manuals are available at:**

- http://www.turnerdesigns.com/products/submersible-fluorometer/c3-submersible-fluorometer

**Liquid Robotics manuals are available on the documentation CD included with each Wave Glider and by contacting the Liquid Robotics Support Organization at:**

- Telephone:

  (US) 888-574-4574

  (International) 1-408-636-4205

- Email: support@liquidr.com

  Before contacting LRI support, please have available the serial number of the vehicle(s) or part(s) that you are contacting LRI Customer Support about.

# Chapter 1. System Overview

## Product Overview

The integration of the C3 sensor into the LRI Wave Glider® SV2 provides a mobile, long endurance platform for measuring various phenomena found in water.

The C3 sensor takes measurements by looking at reflected or fluorescent responses to various wavelengths of light. A C3 sensor is a single module with three different fluorometer sensors (referred to as C1, C2, and C3). The wavelength used by a particular sensor determines the material being measured.

The C3 sensor takes repeated measurements from the three fluorometers and reports values in ASCII strings over the serial port. The measurement data is then correlated with the GPS location and UTC time from the vehicle. The data from the sensor can be collected as fast as 1 sample per second (a rate that can easily overwhelm the available Iridium SBD bandwidth) down to as slow as 1 sample per hour (3600 seconds).

Depending on the configuration of the optical units, a C3 sensor can detect any combination of three of the following materials:

- Blue-GreenAlgae (Cyanobacteria) [http://www.turnerdesigns.com/applications/cyanobacteria-phycoerythrin-and-phycocyanin]
  - Phycocyanin (Freshwater)
  - Phycoerythrin (Marine)
- CDOM/FDOM [http://www.turnerdesigns.com/applications/colored-dissolved-organic-matter-cdom]
- Chlorophyll *In Vivo* [http://www.turnerdesigns.com/applications/chlorophyll]
- Fluorescent Dye Tracing [http://www.turnerdesigns.com/applications/fluorescent-dye-tracing]
  - Fluorescein
  - PTSA
  - Rhodamine WT
- Hydrocarbons [http://www.turnerdesigns.com/applications/oil-and-fuel-in-water]
  - Crude Oil
  - Refined Fuels
- Turbidity [http://www.turnerdesigns.com/applications/turbidity]
- Wastewater Monitoring [http://www.turnerdesigns.com/applications/wastewater-monitoring-fluorometer-applications]
  - Optical Brighteners

– Tryptophan

The LRI C3 sensor management program (C3 program) resides on an LRI Sensor Management Computer (SMC). When executed, the C3 program creates a C3 process that manages the serial connection to the C3 sensor and exports data to other applications and shore-side users.

The C3 sensor is powered by the Wave Glider’s solar energy based electrical system. By default, the C3 program processes C3 data and transmits it to WGMS. Samples are stored persistently on-board the Wave Glider in their native formats with associated GPS tags containing the latitude, longitude, date, and time for each sample. Custom post-processing applications written in C, C++, Java, and a variety of scripting languages can run on-board in the Linux-based SMC. Data can be sent to shore using standard Wave Glider SV2 communications channels as well as payload-resident modems.

You do not need to configure the C3 sensor using the C-Soft program provided by Turner Designs before installing the C3 sensor. Instead, the C3 sensor can be configured when the C3 program is running. Note that the C3 program will ignore the data logging mode if the sensor has been pre-configured using the Turner C-Soft program. As a result, the C3 sensor will not automatically emit data after power-on, instead you will need to send a start command to receive data from the sensor.

# Mechanical Architecture

The C3 sensor body is installed in a Wave Glider Float payload bay where it can be in contact with water. Related electronics are installed in a Wave Glider SV2 payload dry box. The payload must go through an LRI standard installation process to maintain pressure and avoid water leakage.


## C3 Sensor Physical Specifications

| | |
|---|---|
| Weight in Air | 1.64 kg; 3.6 lbs |
| Length | 23 cm; 9.1 in |
| Diameter | 10 cm; 3.9 in |
| Material | Delrin Plastic |
| Temperature | -2 to 50 degrees C. |
| Depth | 0 to 600 meters |

# Fluorometer Specifications

| Application | Minimum Detection Limit | Dynamic Range |
|---|---|---|
| CDOM/FDOM | 0.15 ppb** 0.5 ppb*** | 0-1250 ppb** 0-5000 ppb*** |
| Chlorophyll *in vivo* | 0.025 µg/L | 0-500 µg/L |
| Fluorescein Dye | 0.01 ppb | 0-500 ppb |
| Oil - Crude | 0.2 ppb*** | 0-2700 ppb*** |
| Oil - Fine | 10 ppb* 10 ppm**** | >10,000 ppb* >100 ppm**** |
| Optical Brighteners | 0.6 ppb*** | 0-15,000 ppb*** |
| Phycocyanin<br>(Freshwater Cyanobacteria) | 2 ppbPC | 0-40,000 ppbPC |
| Phycoerythrin<br>(Marine Cyanobacteria) | 0.15 ppbPE | 0-750 ppbPE |
| PTSA Dye | 0.1 ppb*** | 0-650 ppb*** |
| Rhodamine Dye | 0.01 ppb | 0-1000 ppb |
| Tryptophan | 3 ppb | >20,000 ppb |
| Turbidity | 0.05 NTU | 0-3000 NTU |

* 1,5 Napthalene Disulfonic Disodium Salt

** Quinine Sulfate

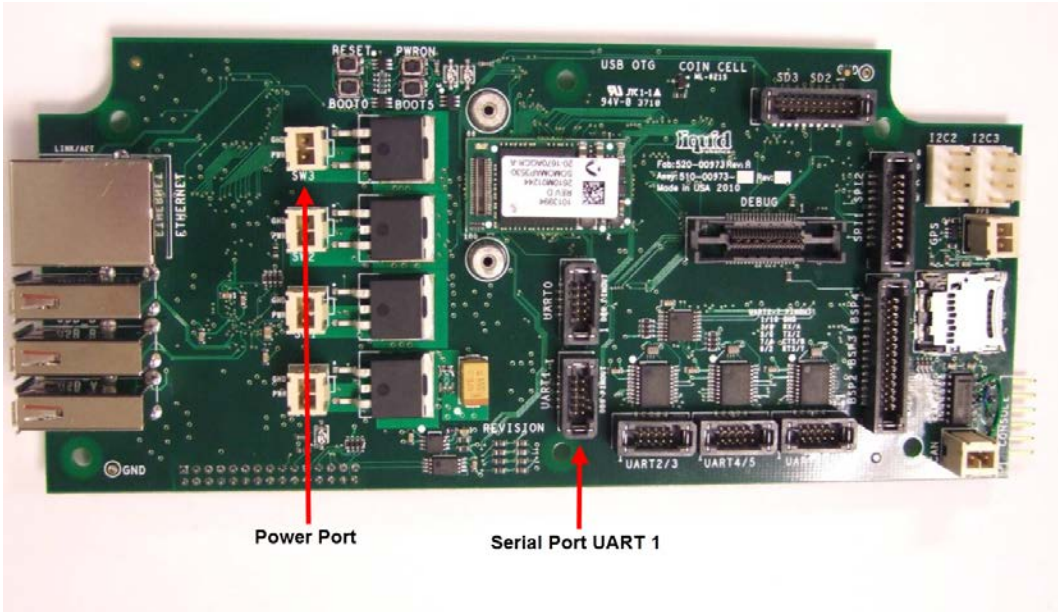
*** PTSA (1,3, 6, 8 - Pyrenetetrasulfonic Acid Tetrasodium Salt)

**** BTEX (Benzene, Toluene, Ethylbenzene, Xylenes), PC Phycocyanin pigment from Prozyme diluted in Deionized water, PE Phycoerythrin pigment from Prozyme diluted in Deionized water

# Electronic Architecture

The C3 sensor is controlled by the C3 program running on the SMC. Power is provided by one of the power output ports on the SMC. The RS-232 connector on the C3 sensor is connected to a serial port that has a DTR line. Only UART0 and UART1 have a DTR line. LRI recommends using UART1.

**Figure 1.1. System Management Computer (SMC)**




Cables are included with the product when the integration is performed by Liquid Robotics.

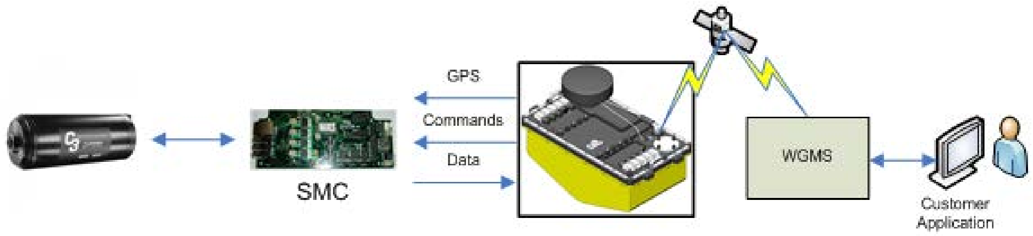
# Electrical Specifications

| | |
|---|---|
| Output | Digital (ASCII); Analog (0 to 5 volt) - optional |
| Interface | RS232 Interface |
| Minimum Sample Interval | 1 second |
| Minimum Power Supply | 8 to 30 volts; 5 watts |
| Maximum Current Draw at 12 volts | - operational: 200 mA<br>- sleep: 3 mA |

# Software Architecture

Broadly, data flows through the paths shown in Figure 1.2.

**Figure 1.2. C3 Sensor Control and Data Flow**



Use this document to plan your use of the C3 sensor. Once you have established your plan, you will use the Wave Glider Management System (WGMS) to send commands to the C3 program running on the SMC. These commands set parameters such as the data collection frequency, and provide control functions such as power-on/off, start, or stop. Commands can also be issued from a shell script running on the SMC, using the LRI ServerExec command.

When commanded by the user, the C3 sensor sends samples to the C3 program. The C3 program processes the ASCII strings and stores the samples in the SMC file system.

The SMC event manager delivers the data to other applications that request it. There are two types of events. A "C3SampleHeader" event contains header information for the C3 sensor and a "C3SampleData" event contains data samples. Typically the data is delivered to the SMC-resident `PayloadInterface` program and saved in the SMC file system. `PayloadInterface` packages the data into an LRI message format and sends it through the Wave Glider SV2 Command and Control (C&C) communications infrastructure for delivery to WGMS where the data is also stored in the WGMS database.

Applications other than the C3 program can also subscribe to the "C3SampleData event to receive data. For example, a custom program could filter and compress samples and deliver them to the payload manager to be forwarded to WGMS. This could reduce data transmission time, saving power and bandwidth.

# Chapter 2. Getting Started with the C3 sensor

## Mission Planning

The combination of the Wave Glider SV2 sensor management system with the highly configurable C3 program supports a large number of possible missions. When developing a detailed mission plan, you must balance mission objectives and environmental constraints.

One of the key advantages of the Wave Glider SV2/C3 sensor integration is that it is easy to modify a mission at any time, including when the Wave Glider is off-shore. The settings on the C3 sensor can be controlled or reconfigured using WGMS. For more information see Chapter 4: *Commands*.

When you configure the system, key considerations include:

- **Power Budget** – The C3 sensor is a relatively low-power sensor. However, its power consumption must be considered when assessing the total power budget for the Wave Glider SV2 system.
- **Communications Bandwidth** – It is possible to configure the system to generate more raw data than the Iridium communications channel can transport. Once a session is established, the underlying maximum data rate of the Iridium network is 2400 bps. If those options are not appropriate for your application, it is possible to write custom code for the SMC to re-parse the data into a format that requires less bandwidth to transport.

  Note that communications requirements should be included in the power budget – when transferring at its maximum data rate, an Iridium 9522B can consume more than 2 watts.
- **Scripts** – If you need to send more than a few commands during the mission, consider incorporating those commands into a script. A script offers the flexibility to support several different data collection and reporting schemes. You should confirm that the scripts you develop work properly before running them at sea.
- Onboard storage – The C3 program stores raw data samples in the SMC file system. The SMC generally ships with a 16GB Secure Digital flash file system, of which approximately 12GB is available for user data. You should estimate how much data you expect to generate and configure the system accordingly. Managing file system space and deleting unneeded files is the mission operator's responsibility.

## Setting Up the C3 Program and the C3 Sensor for a Mission

This section assumes that the C3 subsystem, including the C3 program and related configuration files and scripts on the SMC, are installed and fully tested.

The C3-config.xml file stores the C3 program settings. To fully control the C3 sensor, the C3 program toggles the DTR line in the first 5 seconds after being powered on - the C3 sensor must be connected to a UART port with a DTR line. If UART1 is selected, `<portName>/dev/ttyLX1</portName>` should be specified in the xml file.

All sensors on the same SMC will share the same SMC board IDs, but they will have unique task IDs. The task ID for the C3 sensor is 23 or 0x17. If the SMC has the board ID of 0x20, the board ID and the task ID concatenation yields the address of 8215 or 0x2017. You can find this specification in the `C3-config.xml` file as follows:

```
<payloadDescription class_id="12" tracking_level="0" version="0">
         <boardID>255</boardID>
         <taskID>23</taskID>
         <description>C3 sensor interface</description>
         <ior>(This will be overwritten)</ior>
         <readableName>C3</readableName>
</payloadDescription>
```

### Note

255 is the common board ID of the SMC, identifiable by `PayloadInterface`.

To make sure the C3 sensor always starts properly, the C3 program completes the following steps:

1. Powers off the C3 sensor.
2. Sets the DTR line.
3. Powers on the C3 sensor.
4. Clears the DTR line after 5 seconds.
5. Sleeps for another 40 seconds allowing the C3 sensor firmware to complete its initialization.
6. The C3 program reads the C3 sensor configuration, then sends out the first start command followed by a stop command after receiving a couple of initial samples.

This start/stop sequence is necessary to leave the sensor in the pause mode rather than in the sleep mode. This way, the C3 sensor is always ready to receive a command. The start/stop sequence completes the initialization process, which takes about 60 seconds.

# Maintenance and Service

Before a mission, you should perform a basic blank reading on the sensor by cleaning it, getting a clear water reading with deionized water, and then comparing the reading to the most recent data from the lab. This procedure requires filling a flow cap with deionized water and fitting the cap to the instrument in the field. Record data for about a minute or so, then compare the data with the most recent calibration sheet.

LRI recommends having the C3 sensor fully re-calibrated once each year, or after 2 months of cumulative operating time. However, if the C3 sensor continues to gather reasonable data, you could go longer before re-calibration.

# Chapter 3. Deployment Checklist

LRI recommends that you take the following actions before launching a mission:

- Prepare a detailed mission plan.
- Calibrate the C3 sensor. See “Maintenance and Service” (page 7).
- Confirm that enough storage is available on the SMC file system.
- Verify that the data frequency is correct by looking at the processed data in the `c3raw.csv` file.
- Using WGMS, confirm that the C3 samples appear in the Fluorometer Records you can access from the Sensor Data menu in the WGMS Vehicles screen. This ensures that you have electrical connectivity between the Wave Glider SV2 and WGMS.

# Chapter 4. Commands

# C3 Commands

The C3 commands invoke operations that directly control the C3 sensor. Commands start with the key word “C3” followed by a required command parameter. The commands described in this chapter are for the C3 program that runs on a System Management Computer (SMC). If you have a legacy Wave Glider SV2 vehicle with an LRI integrated Turner C3 sensor, see Appendix B: *PIB C3 Sensor Program Commands* for information on the commands for the C3 sensor-specific application that runs on the PIB.

## Sending Commands to the SMC from WGMS

To send C3 commands to the SMC:

1. Select the Wave Glider vehicle you want to issue a command to from the drop-down menu in the upper left-hand corner of the WGMS dashboard.
2. Click on **More Commands...** located just below the command buttons on the left side of the WGMS dashboard - a **Simple Command Form** will appear with the **Vehicle Command Info** tab open.
3. Confirm that the correct vehicle name appears in the **Vehicle** field.
4. Enter a reason for issuing the command in the **Command Reason** field.
5. Select **Write Console Script** in the **Command Type** drop-down menu.
6. Enter 0 in the **Function** field.
7. Enter the SMC board address in the **Board Address** field. The task ID for the Turner C3 is 0x17, so by convention the board address is 0x2017 (8215) for the aft payload, 0x2117 (8471) for the forward payload, or 0x3017 (12311) for a towfish payload (not currently available). If your configuration does not follow this convention, see “Determining the Board Address” (page 9).
8. Select the communication channel WGMS will use to send the command to the Wave Glider in the **Com Channel** drop-down menu. The default communications channel will be displayed. If any other communications channels available to the vehicle will be listed in the **Com Channel** drop-down menu.
9. Enter the command you want to issue in the **Script Buffer** field of the **Simple Command Form**.
10. Click **Send** at the bottom of the **Simple Command Form** to issue the command.

### Determining the Board Address

To issue a command to the SMC, the decimal version of the combined Board ID and Task ID for the SMC that the C3 is connected to must be entered into the **Board Address** field in the **Vehicle Command Info** tab of the **Simple Command Form**.

The combined Board ID and Task ID consists of two hexidecimal values:

- The first value is the Board ID for the SMC you want to issue a command to. The Board ID will be between0x00 and 0xFF -exclusive.
- The second value is the Task ID. The Task ID for a C3 is 23.

### Note

If you do not know the Board ID for the SMC, see "Obtaining the Board ID" (page 10).

**Examples:**

The combined Board ID/Task ID for a SMC with a Board ID of 0x20 would be 0x2000. To determine the decimal version, multiply the Board ID value by 256 and add the Task ID {(32)(256) + 00}. The result is 8192. This is the number you would enter into the Board Address field on the Vehicle Command tab.

If the Board ID for the SMC is 0x21, and the command is directed to the C3, the combined hexidecimal values would be 0x2117. To determine the decimal version, multiply the Board ID value by 256 and add the Task ID {(33)(256) + 23}. The result is 8471. This is the number you would enter into the **Board Address** field on the **Vehicle Command** tab.

## Obtaining the Board ID

The Board ID for the SMC that the C3 is associated with should be included in the Mission Plan (the Task ID for a C3 is 23).

The Board ID for the SMC can also be obtained by issuing the **FCC -s** command to Board Address **0** provided:

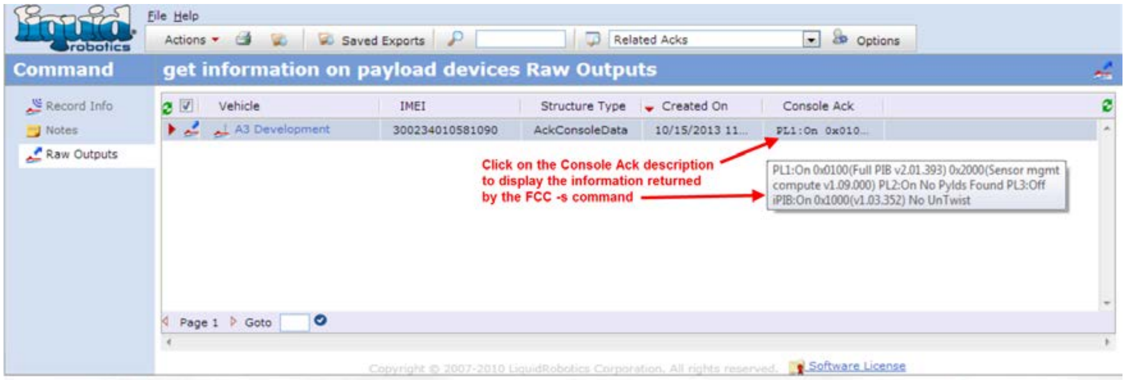
- The payload is turned on (if the payload is not turned on, see "Turning a Payload On or Off" (page 11).
- Float C&C Firmware version 2.0.995 (LRI Part Number 00655-09) or later is installed on the C&C.

The **FCC -s** command will return the power status of each payload port, and the Board ID and Task ID for each enumerated device on each payload port.

When the **FCC -s** command is "ACKed", double click on the **Command Reason** in the **Commands** list at the bottom of the WGMS dashboard - a **Simple Command Form** will appear.

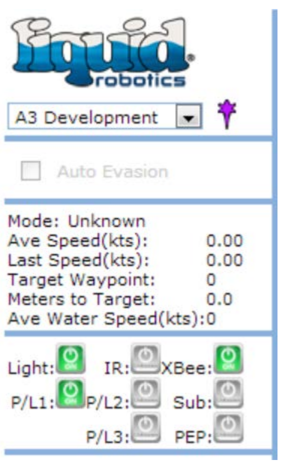
Select **Command Form** in the drop-down menu at the top of the **Simple Command Form** to display the **Command Form**, then click on **Raw Ouputs** and wait for the vehicle to respond.

Click on the information displayed in the **Console Ack** column of the raw output to display the power on/off status of each payload port, and the Board ID and Task ID for each enumerated device on each payload port.




## Turning a Payload On or Off

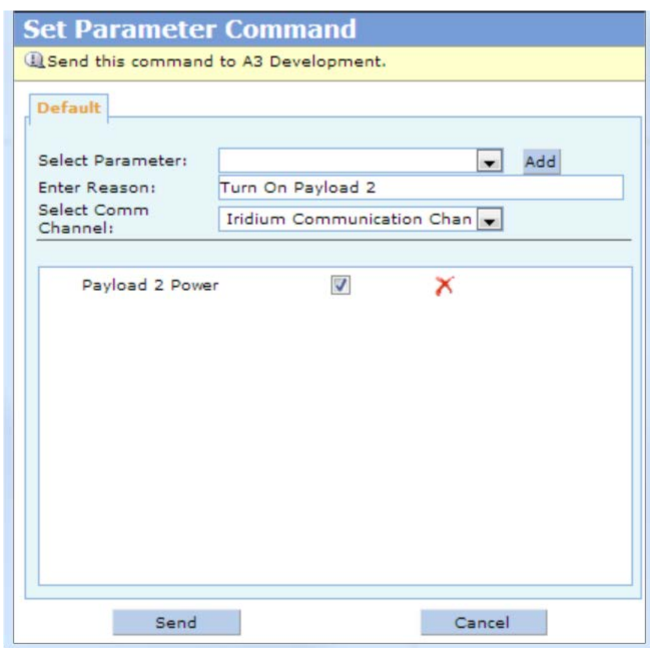
The P/L1, P/L2, and P/L3 payload buttons on the WGMS dashboard turn green when a payload power on command is issued, and they turn gray when a payload power off command is issued. In the following screen capture, Payload 1 is turned on and Payloads 2 and 3 are turned off.




Until telemetry is returned from the Wave Glider indicating that a payload actually is On or Off, the color of these buttons alone is not a reliable indicator of payload power status. See “Payload On Command Sequence” (page 13) for information on circumstances where the payload buttons on the WGMS dashboard may not accurately reflect the On/Off status of a payload.

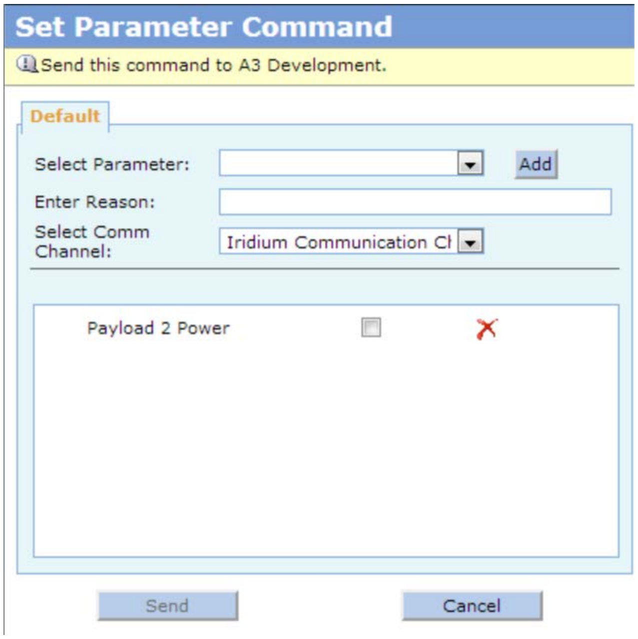
To turn a payload on, click the gray button associated with that payload and a **Set Parameter Command** dialog box will appear with the parameter field pre-populated.


Enter a reason in the **Enter Reason:** field, and select a different communications channel in the **Select Comm Channel:** field (if you are not using Iridium), then click the cursor in an unpopulated area of the parameter field to activate the **Send** button.

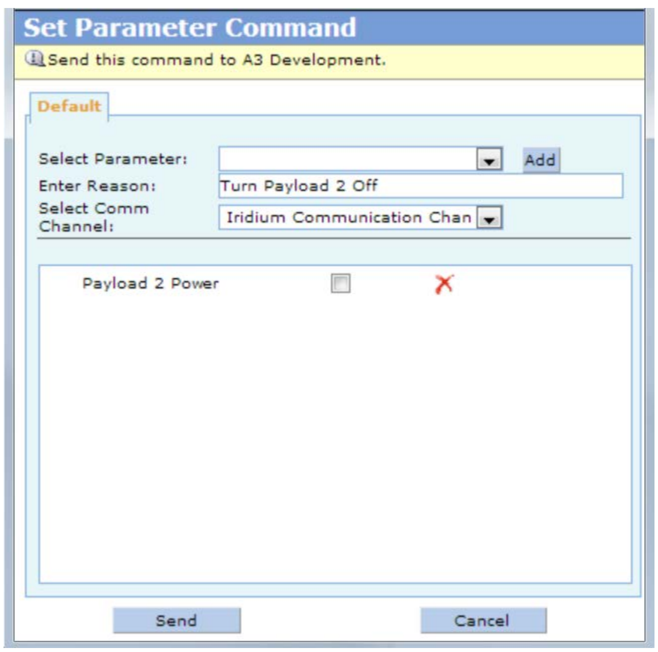



Click **Send** to turn the payload on - the button associated with that payload should turn green.

To turn a payload off, click the green payload button associated with that payload and a **Set Parameter Command** dialog box will appear with the parameter field pre-populated.




Enter a reason in the **Enter Reason:** field, and select a different communications channel in the **Select Comm Channel:** field (if you are not using Iridium), then click the cursor in an unpopulated area of the parameter field to activate the **Send** button.




Click **Send** to turn the payload off - the button associated with that payload should turn gray.

## Payload On Command Sequence

When turning a payload on you must allow time for Iridium communications latency (telemetry intervals), and for the payload to boot, before issuing any other commands to the payload. Figure 4.1 shows the sequence of events from the time that you click a **P/L** button on the WGMS dashboard to issue a payload power on command, to the time when telemetry is returned from the Wave Glider C&C indicating that the payload is actually powered on.

The time between when a payload on command is issued, and when a payload actually is on, depends on the length of the telemetry interval and on the WGMS "Payload Power On Enumeration Delay" setting.

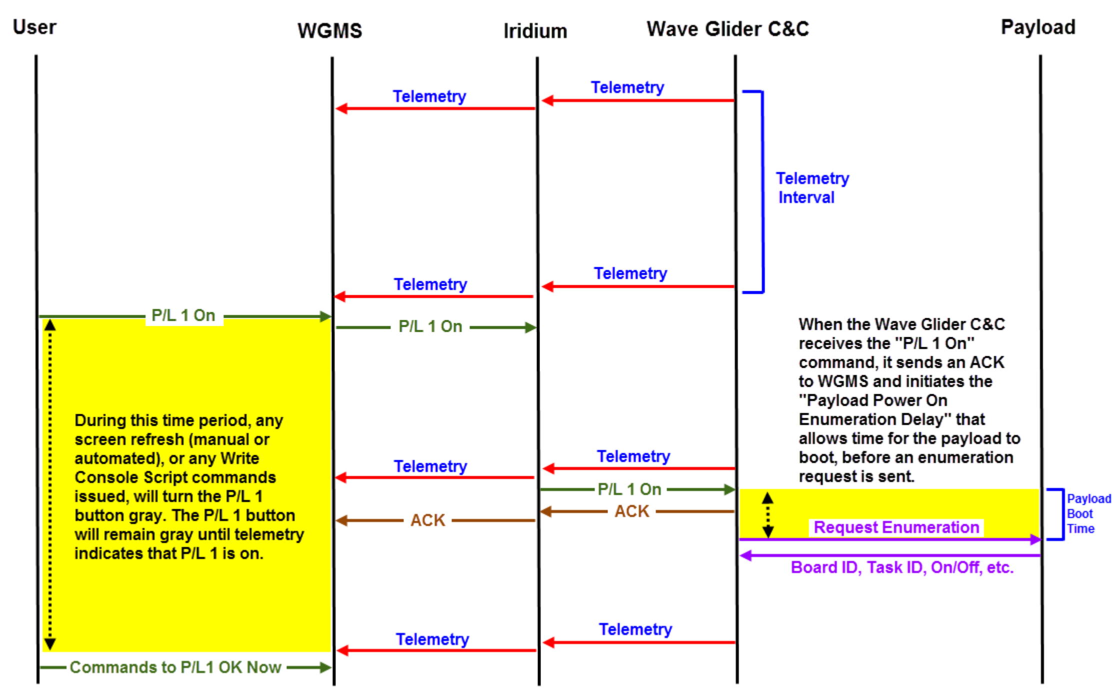
- Telemetry Interval - The time between each telemetry broadcast from the Wave Glider C&C. The telemetry interval is determined by the WGMS **Set Status** command - it can range from 1 minute to 1440 minutes in 1 minute increments. When the Iridium communications channel is used, the telemetry interval is typically set to 5 minutes.

- Payload Power On Enumeration Delay - The time between when a payload is powered on and when the Wave Glider C&C issues an enumeration request to the payload. The purpose of this setting is to allow enough time for payloads to boot so they are able to respond to an enumeration request. The Payload Power On Enumeration Delay is determined by the WGMS **Set Parameters** command. The default setting is 15 seconds, but it can range from 1 to 32767 seconds. This setting applies to all of the payloads on a Wave Glider and should allow enough time for the payload with the longest boot sequence.

When a **P/L** button is clicked to turn a payload on, the **P/L** button instantly turns green. Until telemetry is returned indicating that the payload is actually powered on, a screen refresh will cause the **P/L** button color to revert to gray. Once the payload is actually powered on and the telemetry indicating the payload power on status has been received by WGMS, a green **P/L** button is a reliable indicator that the payload is powered on. Note, an ACK for the payload power on command confirms that the Wave Glider C&C received the command and issued it to the payload - it does not confirm that the payload has been powered on.

It is important to wait until telemetry confirms that a payload is powered on, before you attempt to issue any commands to that payload.

### Figure 4.1. Payload Power On Command Sequence



# Command Syntax

### Note

**All C3 commands are case-insensitive. Upper case characters are used in command syntax and examples for emphasis only.**

### Table 4.1. Command Syntax

| | |
|---|---|
| **Bold** | Bold items in the command syntax must entered. |
| *Italic* | Italic items in the syntax are place holders for information that you must include when you enter the command. Italic items are not to be taken literally - do not include them when you enter a command. |
| Vertical Bar "\|" | Vertical bars (also known as "pipes") are used to separate the short form and the long form of commands, or to separate various command options. Vertical bars are not to be taken literally - do not include them when you enter a command. |
| S p a c e s | All spaces should be taken literally. If a command's syntax has a space, use that space when entering the command. |
| EXAMPLES: | |
| The syntax for the C3 off command is: **C3 ---off \| -f**<br><br>The long form of the command is: **C3 --off** The short form is: **C3 -f** | |
| The syntax for the C3 setconfig command is: **C3 --setconfig** ***`params values`***<br><br>At least one parameter and its associated value must be entered. The C3 setconfig command for taking 100 samples with a 30 millisecond power delay is: **C3 --setconfig nsamples 100 pwrondelay 30** | |
| The SELECT command syntax is: **SELECT --nsample \| -n \| --time \| -t \| --distance \| -d** ***`N`***<br><br>Only one of the three parameters and its associated value can be entered. The short form of the SELECT command for taking 100 samples is: **SELECT -n 100** | |

**Synopsis**

**C3 —-on | -o**

**C3 —-off | -f**

**C3 --start**

**C3 --stop**

**C3 --getconfig**

**C3 --setconfig** ***`params values`***

**C3 --status**

**C3 --gettime**

**C3 --settime**

**C3 --getcalib *`channel`***

**C3 --getid**

**C3 --run *`pathname`***

# C3 ON

**Syntax**

**C3 --on | -o**

**Summary**

Turn power on.

**Semantics**

Turn the SMC resident power switch on. The C3 controller goes through the initialization for about 60 seconds and ends at the stop state where it is ready to receive a new command.

To make sure the C3 sensor always starts clean, the C3 controller powers off the sensor and sets the DTR line. It then powers on, clears the DTR line within the first 10 seconds, and sleeps for another 40 seconds to allow the C3 sensor firmware to complete its internal initialization. The C3 controller reads the sensor configuration. It sends out the first start command and then a stop command after receiving a few initial samples. This start/stop sequence is necessary to leave the sensor in the pause mode rather than in the sleep mode. This way, the C3 sensor is always ready to receive a command. The start/stop sequence completes the initialization process, which takes approximately 60 seconds.

### Note

The C3 sensor does not automatically emit data after being powered on unless **autostart** is set to 1 in the `C3-config.xml` file.

**For Example...**

**Command:**

**`ServerExec C3 "C3 --on”`**

**Response:**

```
<root create-time="2012/12/13 22:46:23.439829"start-time="2012/12/13 22:46:23.440256" end-time="2012/12/13
22:47:20.069356" cmd="C3 --on">
<output value="C3 ON"/>
</root>
```

# C3 OFF

**Syntax**

**C3 --off | -f**

**Summary**

Turn power off

**Semantics**

Turn the SMC resident power switch off to cut power to the C3 sensor.

**For Example...**

**Command:**

**`ServerExec C3 "c3 --off"`**

**Response:**


```
<root create-time="2012/12/13 22:30:09.884665" start-time="2012/12/13 22:30:09.885123" end-time="2012/12/13
22:30:09.946342" cmd="c3 --off">
<output value="C3 OFF" />
</root>
```


# C3 START

**Syntax**

**C3 --start**

**Summary**

Receive data from the C3 sensor, format the raw ASCII data, and save the formatted data in the output data file.

**Semantics**

The C3 controller sends a start command with the array of [0x4, 0x1, 0x1] to the C3 sensor. The sensor responds with string of 24 bytes as shown the following example:

```
RX data = 4,20,64,98,64,45,64,94,192,0,192,0,192,0,0,0,154,100,36,245,162,184,7,131
```

| Byte offset | Byte size | Values | Meaning |
|---|---|---|---|
| 0 | 1 | 4 | START response |
| 1 | 1 | 20 | 20 bytes of data |
| 2 | 2 | 64,98 | Data on channel 1 |
| 4 | 2 | 64,45 | Data on channel 2 |
| 6 | 2 | 64,94 | Data on channel 3 |
| 8-13 | 6 | 192,0,192,0,192,0 | Data on empty channels |
| 14 | 2 | 0,0 | Pressure |
| 16 | 2 | 154,100 | Temperature |
| 18 | 4 | 36,245,162,184 | Date and Time |
| 20 | 2 | 7,131 | Checksum |

The C3 controller parses the input ASCII data, converts it to a readable format, and writes the formatted data to the output data file `c3raw.csv` as shown in the following example:

```
Latitude,Longitude,SysDate,SysTime,C3Date,C3Time,Channels(1-6),Pressure,Temperature 37.4108588,-122.0044443,
13-12-2012,23:33:32,12/13/2012,23:33:19,38.8,16.8,38,0,23.55
```

The name of the output data file is set in the `C3-config.xml` file located in **`/home/smc/roots/200_C3/`** The C3 program prepends the formatted data with the GPS coordinates, date, and time for each sample.

**For Example...**

**Command:**

**`ServerExec C3 “C3 --start”`**

**Response:**

```
<root create-time="2012/12/13 23:34:26.692212" start-time="2012/12/13 23:34:26.692639" end-time="2012/12/13
23:34:29.936845" cmd="C3 --start">
<output value="12/13/2012,23:34:16,39.2,18,37.6,0,23.55" />
</root>
```

# C3 STOP

**Syntax**

**C3 --stop**

**Summary**

The C3 sensor stops transmitting data.

**Semantics**

The C3 controller sends a stop command with the a byte array of [0x4, 0x1, 0x0] to the C3 sensor. The sensor responds with either a string of 29 or 5 bytes. If 29 bytes, then the first 24 bytes are the last data string followed by the 5 stop bytes. The last data string is also saved in the output data file.

**For Example...**

**Command:**

**`ServerExec C3 “C3 --stop”`**

**Response:**

```
<root create-time="2012/12/13 23:34:34.898156" start-time="2012/12/13 23:34:34.899957" end-time="2012/12/13
23:34:35.031523" cmd="C3 --stop">
<output value="OK"  />
</root>
```

# C3 GETCONFIG

**Syntax**

### C3 --getconfig

#### Summary

Get the internal configuration of the C3 sensor

#### Semantics

Retrieve the current configuration by sending an array of [0x3, 0x0] to the sensor. This command can only be executed when the sensor is in the stop state; otherwise, the C3 controller returns a warning: “`C3 running! Please STOP first`”. When the C3 sensor receives the byte array, it responds with 26 bytes, for example:

```
RX data = 3,22,0,0,0,1,13,172,1,199,3,3,3,3,3,3,3,3,0,1,1,0,0,0,1,74
```

| Byte offset | Byte size | Values | Meaning |
|---|---|---|---|
| 0 | 1 | 3 | Get configuration response |
| 1 | 1 | 22 | 22 bytes of data |
| 2 | 2 | 0,0 | Sample limit.<br><br>0 = infinite |
| 4 | 2 | 0,1 | Sample Interval<br><br>1 second minimum |
| 6 | 2 | 13,172 | Power-on delay in milli-seconds |
| 8 | 2 | 1,199 | Channels to read.<br><br>bits 0-5 are C1-C6, bit 6 is pressure, bit 7 is temperature |
| 10 | 8 | 3,3,3,3,3,3,3,3 | Gain on each channel.<br><br>0=1x, 1=10x, 2=100x, 3=auto gain |
| 18 | 2 | 0,1 | Average readings for each sample |
| 20 | 1 | 1 | Idle power in between readings.<br><br>0 = off, 1 == on |
| 21 | 1 | 0 | Number of wiping cycle |
| 22 | 2 | 0,0 | Internal checksum |
| 24 | 2 | 1,74 | External checksum |

The C3 controller parses the input data, turns it into a readable format, and returns the reformatted data in the response.

**For Example...**

**Command:**

**`ServerExec C3 "C3 --getconfig"`**

**Response:**


```
<root create-time="2012/12/14 00:50:19.544788" start-time="2012/12/14 00:50:19.545216" end-time=
"2012/12/14 00:50:21.407139" cmd="C3 --getconfig">
<output value="nsamples=0 period=1 pwrondelay=3500 chanmask=455 gainctrl=33333333 nreadings=1 idlepwr=1
nwiper=0" />
</root>
```


**Command:**

**`ServerExec C3 "C3 --getconfig"`**

**Response:**


```
<root create-time="2012/12/14 01:52:44.590427" start-time="2012/12/14 01:52:44.592228" end-time="2012/12/14
01:52:44.592289" cmd="C3 --getconfig">
<output value="C3 running! Please STOP first."  />
</root>
```


# C3 SETCONFIG

**Syntax**

**C3 --setconfig *`params values`***

**Summary**

Reconfigure the C3 sensor. Each token in the command parameter is separated by a space.

**Semantics**

Change one or more parameters in the configuration. The parameter names must be exact while their orders are flexible.

```
C3 --setconfig nsamples N period N pwrondelay N chanmask N gains NNNNNNNN nreadings
N idlepwr N nwiper N
```

```
uint16_t nsamples;       // sample limit. 0 =INFINITE
uint16_t period;         // sample interval in seconds (data frequency). Minimum = 1 sec.
uint16_t pwrondelay;     // power-on delay in milliseconds
uint16_t chanmask;       // bits 0-5 are C1-C6, bit 6 is pressure, bit 7 is temperature
uint8_t gains[8];        // gain on each channel 0=1x, 1=10x,2=100x, 3=auto gain
                         // gains values must be NNNNNNNNuint16_t nreadings;
                         // number of readings for a sample
uint8_tidlepwr;          // 0=off, 1=on in between readings
uint8_tnwiper;           // number of wiping cycle
```

**For Example...**

**Command:**

**`ServerExec C3 "C3 --setconfig period 1"`**

**Response:**

```
<root create-time="2012/12/14 19:16:40.894471" start-time="2012/12/14 19:16:40.894898" end-time="2012/12/14
19:16:44.408098" cmd="c3 --setconfig period 1">
<output value="nsamples=0 period=1 pwrondelay=3500 chanmask=455 gains=33333333
nreadings=1 idlepwr=1 nwiper=0" />
</root>
```


**Command:**

**`ServerExec C3 "c3 --setconfig period 3 nsamples 10000"`**

**Response:**


```
<root create-time="2012/12/14 19:18:59.078565" start-time="2012/12/14 19:18:59.078992" end-time="2012/12/14
19:19:02.579521" cmd="c3 --setconfig period 3 nsamples 10000">
<output value="nsamples=10000 period=3 pwrondelay=3500 chanmask=455 gains=33333333 nreadings=1 idlepwr=1
nwiper=0" />
</root>
```


**Command:**

**`ServerExec C3 "c3 --setconfig nsamples 0 period 1 gains 33333333 nreadings 1"`**

**Response:**


```
<root create-time="2012/12/14 19:21:40.332674" start-time="2012/12/14 19:21:40.333102" end-time="2012/12/14
19:21:43.813296" cmd="c3 --setconfig nsamples 0 period 1 gains 33333333 nreadings 1">
<output value="nsamples=0 period=1 pwrondelay=3500 chanmask=455 gains=33333333 nreadings=1 idlepwr=1
nwiper=0" />
</root>
```


# C3 STATUS

**Syntax**

**C3 --status**

**Summary**

Get the state of the C3 sensor.

**Semantics**

The C3 controller reports the current state of the sensor.

`OFF` = power off, `ON` = on and paused, `RUNNING` = collecting data.

**For Example...**

**Command:**

**`ServerExec C3 “C3 --status”`**

**Response:**


```
<root create-time="2012/12/14 19:27:05.090602" start-time="2012/12/14 19:27:05.093412" end-time="2012/12/14
19:27:05.093442" cmd="c3 --status">
```

```
<output value="C3 OFF" />
</root>
```


**Command:**

**`ServerExec C3 “C3 --status”`**

**Response:**


```
<root create-time="2012/12/14 19:50:08.088009" start-time="2012/12/14 19:50:08.088437" end-time="2012/12/14
19:50:08.088498" cmd="c3 --status">
<output value="C3 ON" />
</root>
```


**Command:**

**`ServerExec C3 “C3 --status”`**

**Response:**


```
<root create-time="2012/12/14 19:26:43.399526" start-time="2012/12/14 19:26:43.403220" end-time="2012/12/14
19:26:43.403281" cmd="c3 --status">
<output value="C3 RUNNING" />
</root>
```


# C3 GET DATE and TIME

**Syntax**

**C3 --gettime**

**Summary**

Retrieve the internal date and time of the C3 sensor

**Semantics**

Retrieve the internal date and time of the C3 sensor.

**For Example...**

**Command:**

**`ServerExec C3 “C3 --gettime”`**

**Response:**


```
<root create-time="2012/12/15 00:09:45.055032" start-time="2012/12/15 00:09:45.055734" end-time="2012/12/15
00:10:41.414180" cmd="c3 --gettime">
<output value="12/15/2012 0:10:27" />
</root>
```


# C3 SET DATE and TIME

**Syntax**

**C3 --settime**

### Summary

Adjust the internal date and time of the C3 sensor using the SMC system date and time.

### Semantics

The C3 controller uses the system date and time to adjust the internal date and time of the sensor.

### For Example...

**Command:**

**`ServerExec C3 “C3 --settime”`**

**Response:**


```
<root create-time="2012/12/15 01:06:23.318478" start-time="2012/12/15 01:06:23.320310" end-time="2012/12/15
01:06:23.424274" cmd="c3 --gettime">
<output value="12/15/2012 1:6:22" />
</root>
```


# C3 GET CALIBRATION SETTING

### Syntax

**C3 --getcalib *channel***

### Summary

Use this command to learn the configuration of each channel in the C3 sensor.

### Semantics

Get the current settings for a single channel in the C3 sensor module. Each channel is associated with one of three C3 sensor options. Channels 1, 2, and 3 are unique IDs for each of the three sensor options.

### For Example...

**Command:**

**`ServerExec C3 “C3 --getcalib 1”`**

**Response:**


```
<root create-time="2012/12/15 00:23:16.120602" start-time="2012/12/15 00:23:16.121060" end-time="2012/12/15
00:23:16.492451" cmd="c3 --getcalib 1">/12/15 01:06:23.424274" cmd="c3 --gettime">
<output value="chan 1 name Turbidity serial RFU blanktime 0 blankgain1 0 blankgain10 0 blankgain100 0 caltime 0
stdvalue 1 stdreading 1 tempcoef 0 temp 0" />
</root>
```


# C3 GET ID

### Syntax

**C3 --getid**

**Summary**

Identify the sensor by retrieving the serial number and the firmware version of the C3 sensor.

**Semantics**

Retrieve the serial number and the firmware version of the C3 sensor.

**For Example...**

**Command:**

**`ServerExec C3 “C3 --getid”`**

**Response:**


```
<root create-time="2012/12/15 00:58:28.008233" start-time="2012/12/15 00:58:28.008691" end-time="2012/12/15
00:58:29.695019" cmd="c3 --getid">
<output value="serial=2300350 fwmajor=2 fwminor=20 sensorbitmask=455 datalog=0 datalogsize=30720
datalogFree=30720 commMajor=1 commMinor=3" />
</root>
```


# C3 RUN

**Syntax**

**C3 --run *`pathname`***

**Summary**

Execute the list of the C3 commands in the sequential order listed in the script

**Semantics**

Run the C3 commands in the script file. The response of the run command includes all the results of the commands in the script.

Pathname specifies the directory path and the script itself. If the directory path is not specified, the run command assumes the file locates at the local directory of the C3 program which is:

**`/home/smc/roots/200_C3/`**

Run this command when the C3 sensor is in the ON state or the PAUSE state.

**For Example...**

**`/home/smc/roots/200_C3/c3script.txt`**

Contents of the c3script.txt:

```
C3 --status

C3 --getconfig

C3 --start
```

**Command:**

**`ServerExec C3 “C3 --run c3script.txt”`**

**Response:**


```
<root create-time="2012/12/14 22:23:03.328878" start-time="2012/12/14 22:23:03.329488" end-time="2012/12/14
22:23:08.280937" cmd="C3 --run c3script.txt">
<children>
<child create-time="2012/12/14 22:23:03.329519" start-time="2012/12/14 22:23:03.329549" end-time="2012/12/14
22:23:03.329580" cmd="C3 --status">
<ouput value="C3 ON" />
</child>
<child create-time="2012/12/14 22:23:03.330710" start-time="2012/12/14 22:23:03.330740" end-time="2012/12/14
22:23:05.028335" cmd="C3 --getconfig">
<output value="nsamples=0 period=1 pwrondelay=3500 chanmask=455 gainctrl=33333333 nreadings=1 idlepwr=1
nwiper=0" />
</child>
<child create-time="2012/12/14 22:23:05.029129" start-time="2012/12/14 22:23:05.029221" end-time="2012/12/14
22:23:08.280235" cmd="C3 --start">
<output value="12/14/2012,22:22:54,38.8,18.4,38,0,23.26" />
</child>
</children>
</root>
```


# SELECT Command

## Note

**The SELECT command is case-insensitive. Upper case characters are used in command syntax and examples for emphasis only.**

**Syntax**

**SELECT --nsample | -n | --time | -t | --distance | -d *N***

**Summary**

Direct the C3 program to filter the samples from the C3 sensor data stream.

**Semantics**

The SELECT command lets you select specific samples based on sample order, time, or distance. While the command options are mutually exclusive, a SELECT command can be entered at any time and at any state. It will change the ***N*** value instantly at the next sample.

The default sample order setting is 1 causing the C3 program to automatically select and export every data sample. Power cycling the C3 sensor does not change the value of the parameter of the select command.

***N*** is an integer that is interpreted in the context of the type option.

**`SELECT sample`**

**`-n, --nsample`** – This parameter specifies the order in which samples are selected. For example, if ***N*** is 5, the C3 program selects and sends out an event at every 5th sample.

If you select a new sample option while the selected sample is running, the counter is reset to 0 and the next sample will be counted from 1.

**Examples**

**Command:**

**`ServerExec C3 "select -n 5"`**

**Response:**


```
<root create-time="2013/01/14 20:33:50.508410" start-time="2013/01/14 20:33:50.508899" end-time="2013/01/14
20:33:50.509021" cmd="select -n 5">
<output value="select -n 5" />
</root>
```


**Command:**

**`ServerExec C3 "select --nsample 20"`**

**Response:**


```
<root create-time="2013/01/14 20:38:52.057977" start-time="2013/01/14 20:38:52.058465" end-time="2013/01/14
20:38:52.058557" cmd="select --nsample 20">
<output value="select --nsample 20" />
</root>
```


**`SELECT time`**

**`-t, --time`** – The ***N*** parameter sets a “tick” rate for a clock measured in seconds. The clock is initialized to zero by the C3 sensor on receipt of the EXPORT command. The first sample after the tick is selected. For example, if ***N*** is 5, there will be ticks at zero, 5, 10, 15, 20 and so forth. If samples are generated every second and ***N*** is 5, every fifth sample will be selected. If samples are generated every 3 seconds and ***N*** is 5, the sample selected will be the one at the 6th second, the one at the 12th second, the one at the 15th second, the one at the 21st second and so forth. If one sample is generated every 60 seconds and ***N*** is 1, every sample will be selected. In that case, only one sample will be selected even though 60 seconds passed. No sample is selected more than once even if multiple ticks intervene.

**Examples**

**Command:**

**`ServerExec C3 "select -t 30"`**

**Response:**


```
<root create-time="2013/01/14 21:19:21.508577" start-time="2013/01/14 21:19:21.509066" end-time="2013/01/14
21:19:21.509310" cmd="select -t 30">
<output value="select -t 30" />
</root>
```


**Command:**

**`ServerExec C3 "select --time 5"`**

**Response:**


```
<root create-time="2013/01/14 21:20:15.157978" start-time="2013/01/14 21:20:15.158527" end-time="2013/01/14
21:20:15.158802" cmd="select --time 5">
<output value="select --time 5" />
</root>
```

**`SELECT distance`**

**`-d, --distance`** – Analogous to the time parameter, the parameter ***N*** sets a "tick" rate for an odometer measured in meters. The odometer is initialized to zero by the C3 sensor on receipt of the EXPORT command. The selected sample is the first one after the odometer passes a tick. For example, if ***N*** is 100, the selected sample will be the first one after the odometer passes 100, 200, 300 and so forth. The distance is calculated using GPS data.

**Command:**

**`ServerExec C3 "select -d 100"`**

**Response:**

```
<root create-time="2013/01/14 20:55:26.553515" start-time="2013/01/14 20:55:26.554003" end-time="2013/01/14
20:55:26.554278" cmd="select -d 100">
<output value="select -d 100" />
</root>
```

**Command:**

**`ServerExec C3 "select --distance 200"`**

**Response:**

```
<root create-time="2013/01/14 20:56:02.880019" start-time="2013/01/14 20:56:02.880507" end-time="2013/01/14
20:56:02.880629" cmd="select --distance 200">
<output value="select --distance 200" />
</root>
```

# EXPORT Command

## Note

**The EXPORT command is case-insensitive. Upper case characters are used in command syntax and examples for emphasis only.**

**Syntax**

**Export --count | -c *N***

**Export --output | -o *pathname***

**Summary**

`-c | --count` - Specify the total samples to be exported.

`-o | --output` - Direct the exported samples to the export file located at *pathname*.

**Semantics**

**`Export count`**

**`-c | --count`** – Set the total count of the exported samples where ***N*** <0 = infinitive (default), 0 = no export, >0 = upper limit. The C3 program is set to export every sample of the C3 sensor infinitely during the C3 program initialization.

If you enter a new export count option, the counter will be reset and the next sample will be counted from 1 again. Power cycling the C3 sensor does not alter the option value of the export command.

**Examples**

**Command:**

**`ServerExec C3 "export -c 50000"`**

**Response:**

```
<root create-time="2013/01/14 21:22:10.232701" start-time="2013/01/14 21:22:10.233189" end-time="2013/01/14
21:22:10.233311" cmd="export -c 50000">
<output value="export -c 50000" />
</root>
```

**Command:**

**`ServerExec C3 "export --count -1"`**

**Response:**

```
<root create-time="2013/01/14 21:23:18.151134" start-time="2013/01/14 21:23:18.151622" end-time="2013/01/14
21:23:18.151714" cmd="export --count -1">
<output value="export --count -1" />
</root>
```

**`EXPORT file`**

**`-o | --output`** – Direct the exported samples to the exported file located at ***`pathname`***. The default location of the export file is: **`/home/smc/roots/200_C3/c3export.csv`** When you specify a new pathname, the next exported sample will be written to the new file immediately.

**Example**

**Command:**

**`ServerExec C3 "EXPORT --output c3EXPORT2.csv"`**

**Response:**

```
<root create-time="2013/01/14 21:36:46.701857" start-time="2013/01/14 21:36:46.702345" end-time="2013/01/14
21:36:46.702437" cmd="EXPORT --output c3EXPORT2.csv">
```

# Chapter 5. Data Formats

There are three C3 data formats:

- ASCII strings (raw data, difficult to read).
- Processed ASCII strings (processed into easy to read format).
- Compressed processed ASCII strings (each processed ASCII string is compressed into a single packet for forwarding to WGMS).

## Processed Data Format

The C3 program processes the fairly compact ASCII string data from the Turner C3 sensor and saves the processed data in the output data file. The output data file is in an Excel format located in the local directory `/home/smc/roots/200_C3/c3raw.cvs` unless another full path is specified differently in the `/home/smc/roots/200_C3/C3-config.xml`. When the output data file reaches 200KB, it is compressed (zipped) and stored under `/mnt/XDATA/C3`. A new file is then created automatically. Each file has the same header describing each data field separated by a comma.

Following is a sample of the `c3raw.csv` file:

```
Latitude,Longitude,SysDate,SysTime,C3Date,C3Time,Channels(1-6),Pressure,Temperature
37.41084,-122.0043992,21-12-2012,18:01:03,12/21/2012,18:01:01,39.2,17.6,38,0,22.5
37.41084,-122.0043992,21-12-2012,18:01:04,12/21/2012,18:01:02,39.2,17.6,38,0,22.5
37.41084,-122.0043992,21-12-2012,18:01:05,12/21/2012,18:01:03,39.2,17.6,38,0,22.5
```

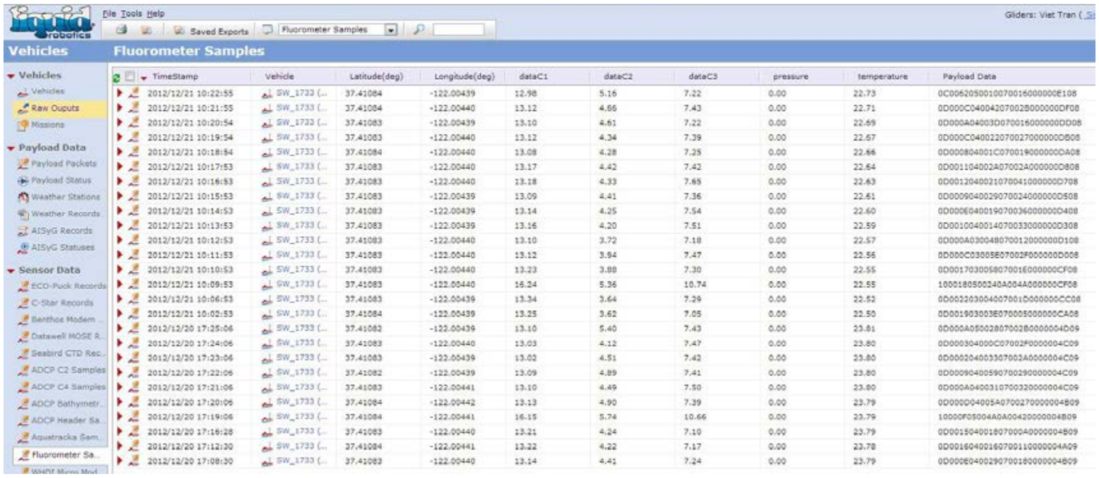
The C3 program prepends the latitude, longitude, system date, and system time in front of the processed data. In the above example, the system time leads the Turner C3 sensor's time by 2 seconds.

Although the C3 program is capable of processing data from 1 to 6 channels, the Turner C3 sensor module is limited by up to 3 channels. You can study the configuration of each sensor/ channel by using the command **"C3 --getcalib N". N = 1, 2, or 3**.

The Turner C3 sensor does not have a pressure sensor. As such, the pressure fields show 0s in the above sample. The temperature values reflect the room temperature of 22.5º C.

## Compressed Data

The data sent to WMGS is compressed to save space in transmission. The data is transmitted in a single telemetry packet that contains all C1, C2, C3, pressure, and temperature values.




When WGMS receives a C3 telemetry packet, it decompresses the packet using the C3 0x91 format. For communication and transmission efficiency, the `PayloadInterface` program accumulates seven C3 telemetry packets and then sends them to WGMS all at once.

# Chapter 6. Telemetry Events

## Sample Header Event

The C3 program broadcasts the header event "C3SampleHeader" at power on, and when you set or change the C3 configuration. The header is a string that defines the current C3 sensor configuration.

```
2013/01/02 18:46:17.753672 dm3730_torpedo [495] [H] properties[C3Event_Header] -> nsamples=0 period=1
pwrondelay=3500 chanmask=455 gains=33333333 nreadings=1 idlepwr=1 nwiper=0/STRING
```

`nsamples` is the sample limit. `0` = infinite.

`period` is the sample interval in seconds. `1` = minimum.

`pwrondelay` is the power on delay in milli-seconds.

`chanmask` with bits 0-5 are C1-C6, bit 6 is pressure, bit 7 is temperature.

`gains` indicates gain factor on each channel. 0=1x, 1=10x, 2=100x, 3=auto gain.

`nreading` is the number of readings for each sample.

`idlepwr` is the idle power in between readings. `0` = off, `1` = on.

`nwiper` is the wiping cycle.

## Sample Data Event

A C3 data event "C3SampleData" consists of a compressed packet containing date, time, C1 value, C2 value, C3 value, pressure and temperature. WGMS accepts compressed data only, other programs subscribing to C3 data events have the option to receive uncompressed data.

This table is the C3 0x91 packet format exchanged between the vehicle and WGMS.

| Field Name | Bytes Used | Byte # | Format |
|---|---|---|---|
| Latitude | 4 | 0-3 | Signed 32-bit integer |
| Longitude | 4 | 4-7 | Signed 32-bit integer |
| Unix Timestamp | 4 | 8-11 | Signed 32-bit integer |
| C1 Integer Value | 2 | 12-13 | Unsigned 16-bit integer |
| C1 Decimal Value | 1 | 14 | Unsigned 8-bit integer |
| C2 Integer Value | 2 | 15-16 | Unsigned 16-bit integer |
| C2 Decimal Value | 1 | 17 | Unsigned 8-bit integer |
| C3 Integer Value | 2 | 18-19 | Unsigned 16-bit integer |
| C3 Decimal Value | 1 | 20 | Unsigned 8-bit integer |

Latitude and Longitude – the location of the float when a C3SampleData packet is received by the SMC (based on GPS information received by the SMC every second from the C&C).

Although the C3 is able to generate GPS data, the C&C GPS data is used because it is more accurate.

Unix Timestamp – the standard Unix timestamp computed from the SMC timestamp. The SMC clock is set from the C&C GPS and should be accurate within a second.

The values for each sensor on the C3 can range from 0.00 up to 65535.99. The values for each sensor on the C3 are assembled by using an integer value on the left side of the decimal point and the decimal value on the right side of the decimal point. For example, if the C1 integer value was 24563 and the C1 decimal value was 75 then the actual value for the C1 fluorometer would be: 24563.75

# Chapter 7. Configuration

The C3 sensor is setup for RS232 communications with the C3 program using the serial port and communications settings specified in the configuration XML file:

`/home/smc/roots/200_C3/C3-config.xml`

# Common XML File Keys to Edit

## For interfacing with the C3 sensor:

**sensorCmdToken** set to default **C3** , for example: “C3 --on"

**serverName** and **payloadDescription** → **readableName** set to default C3, for example: C3 “C3 --on”

Example: ServerExec C3 “C3 --on”

## For controlling the power supply to the C3 sensor:

**powerOnVerb** set default power switch **4** to **1**.

**powerOffVerb** set default power switch **4** to **0**.

## For communicating with the C3 sensor using RS232:

**serialPortSettings** default serial port /dev/ttyLX1, 57600 baud, 8 data bits, 1 stop bit, no parity.

## For non-standard C3 installations including multiple units, the following XML file keys will need to be modified:

serverName

payloadDescription → boardID

payloadDescription → taskID

payloadDescription → readableName

## Auto start

*<autoStart>0</autoStart>* dictates the way the sensor is powered on. The default autoStart setting is 0, which leaves the C3 sensor in the paused mode after initialization. If autoStart is set to 1, the C3 sensor automatically begins producing data when it is powered on.

## Default Data Output File

The following file stores all data on the serial port:

```
<DataOutputFileName>/home/smc/roots/200_C3/c3raw.csv</DataOutputFileName>
```

# C3-config.xml file content:


```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE  boost_serialization>
<boost_serialization signature="serialization::archive" version="9">
<rootlevel class_id="0" tracking_level="0" version="2">
           <sensorProgramSettings class_id="1" tracking_level="0" version="1">
                       <serverName>C3</serverName>
                       <serverKind></serverKind>
                       <softwareVersion class_id="2" tracking_level="0" version="1">
                                   <majorVer>0</majorVer>
                                   <minorVer>0</minorVer>
                                   <subminorVer>1</subminorVer>
                       </softwareVersion>
                       <loggerSettings class_id="3" tracking_level="0" version="0">
                                   <logLevel>LT_DEBUG</logLevel>
                                   <processName>unknown</processName>
                                   <isFileDes>1</isFileDes>
                                   <fileDes>1</fileDes>
                                   <fileName></fileName>
                                   <pipeName></pipeName>
                       </loggerSettings>
                       <commandCompleteEvent class_id="4" tracking_level="0" version="0">
                                 <eventName>CommandComplete</eventName>
                                 <eventType class_id="5" tracking_level="0" version="0">
                                             <typeName>SendCommandCompleteEvent</typeName>
                                             <domainName>LiquidR-SMC</domainName>
                                 </eventType>
                       </commandCompleteEvent>
                       <gpsEvent>
                                 <eventName>GPSSample</eventName>
                                 <eventType>
                                             <typeName>GPSSample</typeName>
                                             <domainName>LiquidR-SMC</domainName>
                                 </eventType>
                       </gpsEvent>
                       <shutdownEvent>
                                 <eventName>Shutdown</eventName>
                                 <eventType>
                                             <typeName>EventBase</typeName>
                                             <domainName>LiquidR-SMC</domainName>
                                 </eventType>
                       </shutdownEvent>
                       <eventTransceiverSettings class_id="6" tracking_level="0" version="2">
                                 <eventServiceName>EventManager</eventServiceName>
                                 <eventServiceKind></eventServiceKind>
                                 <receiveEnable>1</receiveEnable>
                                 <aliasToEvent class_id="7" tracking_level="0" version="0">
                                             <count>0</count>
                                             <item_version>0</item_version>
                                 </aliasToEvent>
                                 <eventFilterEnabled>1</eventFilterEnabled>
                                 <eventNameFilter class_id="9" tracking_level="0" version="0>"
                                             <count>2</count>
                                             <item_version>0</item_version>
                                             <item>GPSSample</item>
                                             <item>Shutdown</item>
                                 </eventNameFilter>
                                 <eventTypeFilter class_id="10" tracking_level="0" version="0">
                                             <count>0</count>
                                             <item_version>0</item_version>
                                 </eventTypeFilter>
                                 <displayReceivedEvents>1</displayReceivedEvents>
                                 <fullyDescribeReceivedEvents>0</fullyDescribeReceivedEvents>
                                 <doNotDisplayList>
```

```
                                <count>0</count>
                                <item_version>0</item_version>
                        </doNotDisplayList>
                </eventTransceiverSettings>
                <payloadManagerClientSettings class_id="11" tracking_level="0" version="0">
                        <serviceName>PayloadManager</serviceName>
                        <serviceKind></serviceKind>
                        <deviceAddedEventName>PayloadDeviceAdded</deviceAddedEventName>
                        <deviceChangedEventName>PayloadDeviceChanged</deviceChangedEventName>
                        <deviceRemovedEventName>PayloadDeviceRemoved</deviceRemovedEventName>
                        <commonEventType>PayloadRegistrationEvent</commonEventType>
                        <commonEventDomain>LiquidR-SMC</commonEventDomain>
                        <payloadDescription class_id="12" tracking_level="0" version="0">
                                <boardID>255</boardID>
                                <taskID>23</taskID>
                                <description>C3 sensor interface </description>
                                <ior>(This will be overwritten)</ior>
                                <readableName>C3</readableName>
                        </payloadDescription>
                </payloadManagerClientSettings>
        </sensorProgramSettings>
        <controllerSettings class_id="13" tracking_level="0" version="1">
                <listenerSettings class_id="14" tracking_level="0" version="0">
                        <maxReadSize>5000</maxReadSize>
                        <intercharacterGapTimeoutMs>10</intercharacterGapTimeoutMs>
                </listenerSettings>
                <serialPortSettings class_id="15" tracking_level="0" version="0">
                        <portName>/dev/ttyLX1</portName>
                        <baudRate>57600</baudRate>
                        <parity>NONE</parity>
                        <characterSize>8</characterSize>
                        <stopBits>1</stopBits>
                        <flowControl>NONE</flowControl>
                        <rts>1</rts>
                        <dtr>1</dtr>
                        <breakTimeoutMultiplier>1</breakTimeoutMultiplier>
                </serialPortSettings>
                <powerOnVerb>sudo SetPowerSwitch 4 1</powerOnVerb>
                <powerOffVerb>sudo SetPowerSwitch 4 0</powerOffVerb>
                <defaultOutfile>c3raw.csv</defaultOutfile>
                <noResponseTries>2</noResponseTries>
                <responseTimeoutSeconds>60</responseTimeoutSeconds>
                <powerOnDelaySeconds>40</powerOnDelaySeconds>
                <outfileSize>204800</outfileSize>
                <defaultTarFile>C3RAW</defaultTarFile>
                <defaultTarDir>/mnt/XDATA/C3</defaultTarDir>
                <defaultExportFile>c3export.csv</defaultExportFile>
                <defaultTarExportFile>C3EXPORT</defaultTarExportFile>
                <autoStart>0</autoStart>
        </controllerSettings>
        <HeaderEventSettings>
                <eventName>C3SampleHeader</eventName>
                <eventType>
                        <typeName>C3Event</typeName>
                        <domainName>LiquidR-SMC</domainName>
                </eventType>
        </HeaderEventSettings>
        <DataEventSettings>
                <eventName>C3SampleData</eventName>
                <eventType>
                        <typeName>C3Event</typeName>
                        <domainName>LiquidR-SMC</domainName>
                </eventType>
        </DataEventSettings>
</rootlevel>
</boost_serialization>
```

# Chapter 8. C3 Custom Reports

C3 custom data reports can be generated and viewed from the Wave Glider Management System (WGMS) dashboard. WGMS Custom Reports are available for a wide range of Wave Glider integrated sensors.

## Viewing Custom Reports

Use the **Custom Reports** screen to, view data from vehicle operations. A Custom Report displays data from a specific payload on a specific vehicle.

The following browsers are supported for viewing WGMS Custom Reports:

| **Browser** | **Version** | **Platform** | **Display** |
|---|---|---|---|
| Internet Explorer | 9, 10 | Windows | Desktop |
| Firefox | 20.0.1 (or later) | Windows | Desktop |
| Safari | 6.0.4 (or later) | Mac | Desktop |
| Safari | (latest) | iPad (iOS 6.1 or later) | Mobile |
| Chrome | 26.0.1410.64 m (or later) | Windows | Desktop |
| Chrome | (latest) | Android (4.0 or later) | Mobile |

There are Custom Reports available for the following Wave Glider integrated payloads:

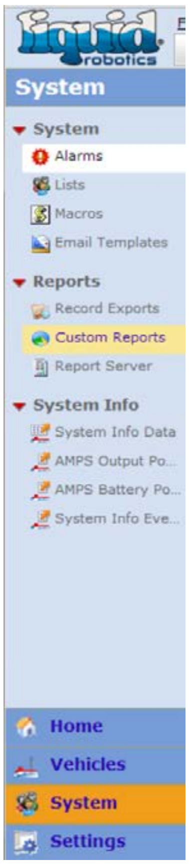
- **ADCP C2**
- **ADCP C4**
- **ADCP Bathymetry**
- **ADCP Header**
- **Aquatracka**
- **Benthos Modem**
- **C-Star**
- **Datawell MOSE**
- **Eco-Puck**
- **Fluorometer**
- **Fluorometer Header**
- **LISST Sensor**
- **pCO2 Sensor**
- **Seabird CTD**

- **VEMCO VR2C**
- **WHOI Micro Modem**

To view Custom Reports:

1. Click on **[Show System Menus]** to open the WGMS system menus.
2. Click on the **System** button to open the **System** screen.
3. Click on the **Custom Reports** side tab in the **Reports** menu on the side-bar.

### Figure 8.1. Opening the Custom Reports Screen



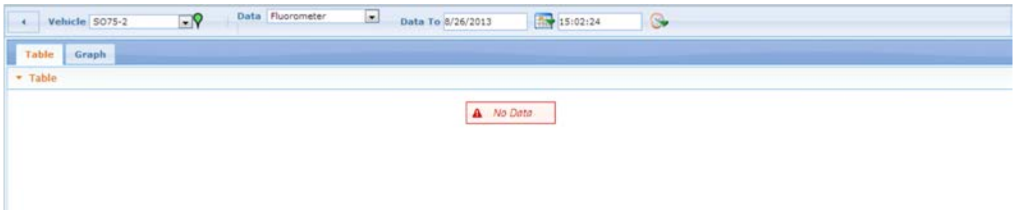

The Custom Reports screen for the selected vehicle will appear.

The toolbar at the top of the Custom Reports screen includes a drop down menu that displays the name of the vehicle that the data in the custom report comes from. If you want to view data from a different vehicle, you can select one from the drop down menu.

Data can be viewed in either of two formats, depending on which tab is selected:

- **Table** - This data format appears when the **Table** tab is selected. By default, the Custom Reports screen opens with the **Table** tab selected displaying data in a tabular format.
- **Graph** - This data format appears when the **Graph** tab is selected displaying data in a a variety of graphs depending on the variables you select.

### Figure 8.2. Custom Reports Screen - No Data Available



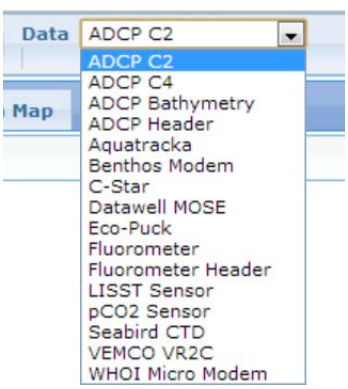

The data type in the **Data** drop-down menu will vary depending on what data type was last viewed in the Custom Reports screen. In Figure 8.2, the Custom Reports screen opened with **Fluorometer** selected in the **Data** drop-down menu, but the No Data icon indicates that the selected vehicle in the **Vehicle** drop-down menu (**SO75-2**) is not generating Fluorometer data. This is the expected behavior for the Custom Reports screen. If the selected vehicle does not have a payload that generates the selected data type, the **No Data** icon will appear, instead of data.

## Note

The Custom Reports screens for all of the supported data types have a similar format, except for the ADCP C2 and ADCP C4 data types. For information on viewing the ADCP C2 and C4 data types, see the *Wave Glider® SV2 Integrated Sensor User Guide* for the Teledyne RD Instruments Workhorse Monitor ADCP.

If you know the type of data that you want to view, and you know the selected vehicle is capable of generating that data type, select it from the **Data** drop down menu.

### Figure 8.3. Data Drop-Down Menu




If you do not know which data types the selected vehicle is capable of generating, you will need to work your way through the list in the **Data** drop-down menu to determine what data types are available from the selected vehicle.

## Note

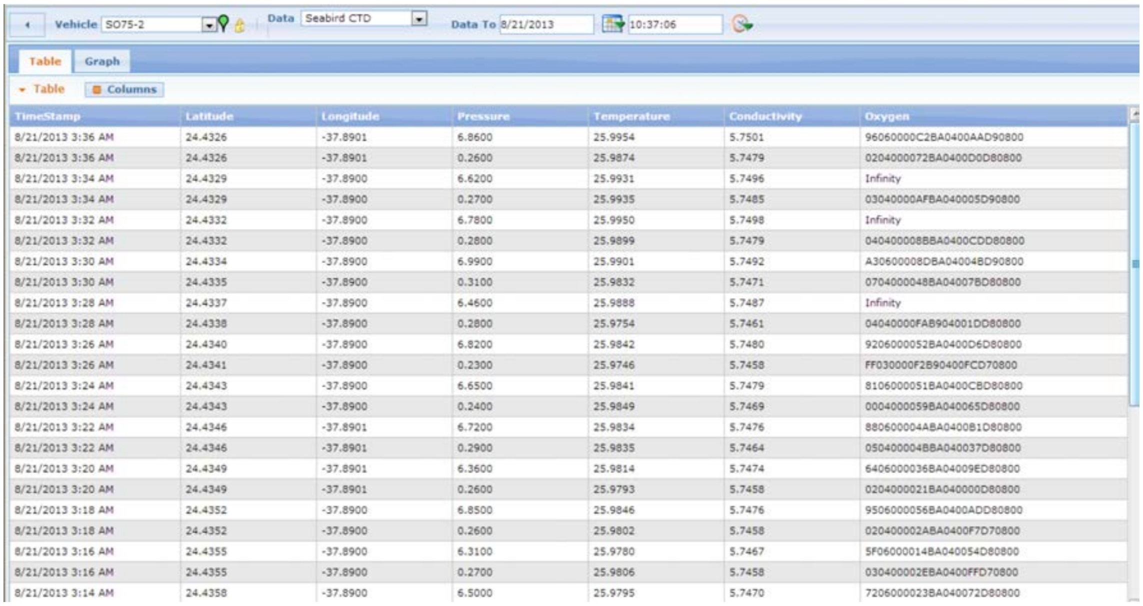
A future WGMS release will automatically populate the **Data** drop-down menu with only the data types that are available from the selected vehicle.

If a data type selected in the **Data** drop-down menu is available for the selected vehicle, an animated loading icon will appear temporarily while the data loads, then the Custom Report screen will display data.

# Custom Reports Table Format

By default, the Custom Reports screen opens to the **Table** tab and displays all of the columns available for a selected data type.

### Figure 8.4. Custom Reports Screen With Data - Table Format




| TimeStamp | Latitude | Longitude | Pressure | Temperature | Conductivity | Oxygen |
|---|---|---|---|---|---|---|
| 8/21/2013 3:36 AM | 24.4326 | -37.8901 | 6.8600 | 25.9954 | 5.7501 | 96060000C2BA0400AAD90800 |
| 8/21/2013 3:36 AM | 24.4326 | -37.8901 | 0.2600 | 25.9874 | 5.7479 | 0204000072BA0400D0D80800 |
| 8/21/2013 3:34 AM | 24.4329 | -37.8900 | 6.6200 | 25.9931 | 5.7496 | Infinity |
| 8/21/2013 3:34 AM | 24.4329 | -37.8900 | 0.2700 | 25.9935 | 5.7485 | 03040000AFBA040005D90800 |
| 8/21/2013 3:32 AM | 24.4332 | -37.8900 | 6.7800 | 25.9950 | 5.7498 | Infinity |
| 8/21/2013 3:32 AM | 24.4332 | -37.8900 | 0.2800 | 25.9899 | 5.7479 | 040400008BBA0400CDD80800 |
| 8/21/2013 3:30 AM | 24.4334 | -37.8900 | 6.9900 | 25.9901 | 5.7492 | A30600008DBA04004BD90800 |
| 8/21/2013 3:30 AM | 24.4335 | -37.8900 | 0.3100 | 25.9832 | 5.7471 | 070400004BBA04007BD80800 |
| 8/21/2013 3:28 AM | 24.4337 | -37.8900 | 6.4600 | 25.9888 | 5.7487 | Infinity |
| 8/21/2013 3:28 AM | 24.4338 | -37.8900 | 0.2800 | 25.9754 | 5.7461 | 04040000FAB904001DD80800 |
| 8/21/2013 3:26 AM | 24.4340 | -37.8900 | 6.8200 | 25.9842 | 5.7480 | 9206000052BA0400D6D80800 |
| 8/21/2013 3:26 AM | 24.4341 | -37.8900 | 0.2300 | 25.9746 | 5.7458 | FF030000F2B90400FCD70800 |
| 8/21/2013 3:24 AM | 24.4343 | -37.8900 | 6.6500 | 25.9841 | 5.7479 | 8106000051BA0400CBD80800 |
| 8/21/2013 3:24 AM | 24.4343 | -37.8900 | 0.2400 | 25.9849 | 5.7469 | 0004000059BA040065D80800 |
| 8/21/2013 3:22 AM | 24.4346 | -37.8901 | 6.7200 | 25.9834 | 5.7476 | 880600004ABA0400B1D80800 |
| 8/21/2013 3:22 AM | 24.4346 | -37.8901 | 0.2900 | 25.9835 | 5.7464 | 050400004BBA040037D80800 |
| 8/21/2013 3:20 AM | 24.4349 | -37.8901 | 6.3600 | 25.9814 | 5.7474 | 6406000036BA04009ED80800 |
| 8/21/2013 3:20 AM | 24.4349 | -37.8901 | 0.2600 | 25.9793 | 5.7458 | 0204000021BA040000D80800 |
| 8/21/2013 3:18 AM | 24.4352 | -37.8900 | 6.8500 | 25.9846 | 5.7476 | 9506000056BA0400ADD80800 |
| 8/21/2013 3:18 AM | 24.4352 | -37.8900 | 0.2600 | 25.9802 | 5.7458 | 020400002ABA0400F7D70800 |
| 8/21/2013 3:16 AM | 24.4355 | -37.8900 | 6.3100 | 25.9780 | 5.7467 | 5F06000014BA040054D80800 |
| 8/21/2013 3:16 AM | 24.4355 | -37.8900 | 0.2700 | 25.9806 | 5.7458 | 030400002EBA0400FFD70800 |
| 8/21/2013 3:14 AM | 24.4358 | -37.8900 | 6.5000 | 25.9795 | 5.7470 | 7206000023BA040072D80800 |

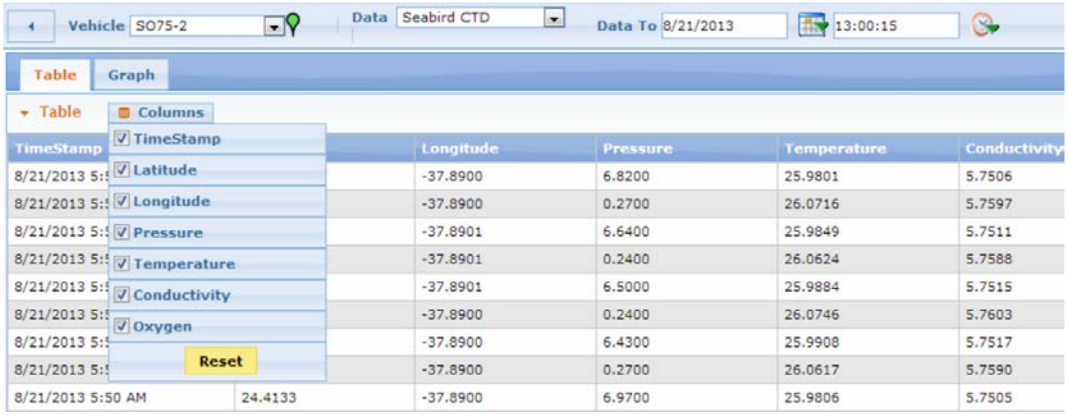
For some data types you will need to scroll to the right to see all of the columns. To limit the number of columns displayed:

1. Click on **Columns** button. A drop-down menu listing the column names will appear.

2. Un-check the check box next to each column name that you do not want displayed and that column will instantly be removed from the Custom Report.

3. To restore all columns, click the **Reset** button.

### Figure 8.5. Changing the Columns that are Displayed




## Note

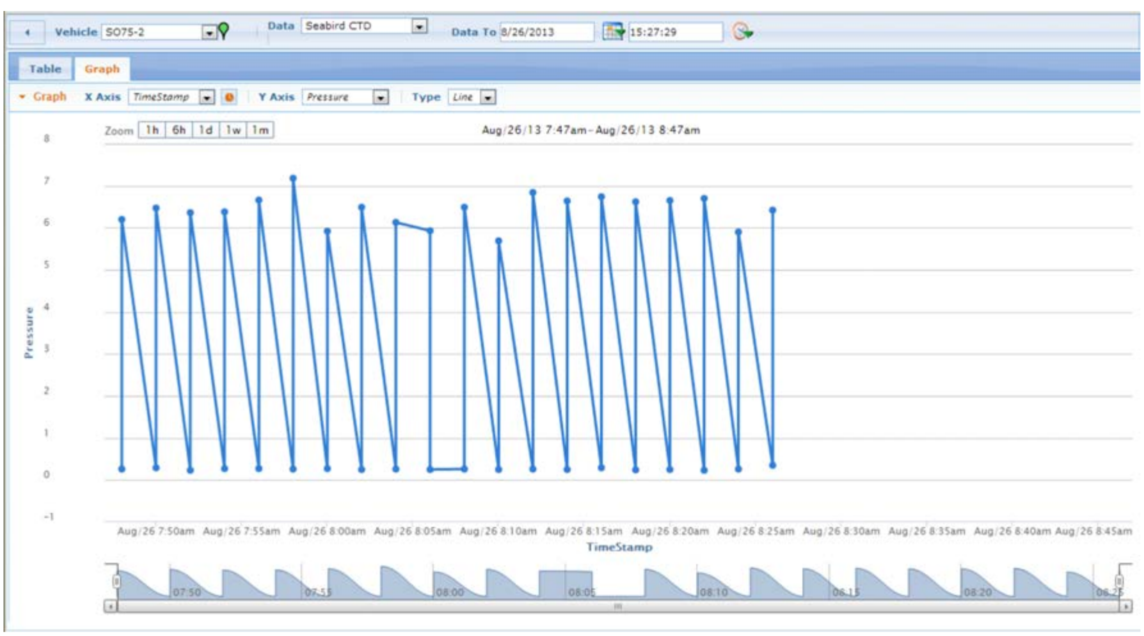
The Custom Report screen provides similar functionality on a tablet, but the navigation is slightly different. When using a tablet, allow extra time for Custom Report screens to generate.

By default, the custom report for a selected data type will display all available data. You can limit the amount of historical data that is displayed by changing the **Data To:** date and time settings. Changing the date and time settings will update the custom report screen that you are viewing.

# Custom Reports Graph Format

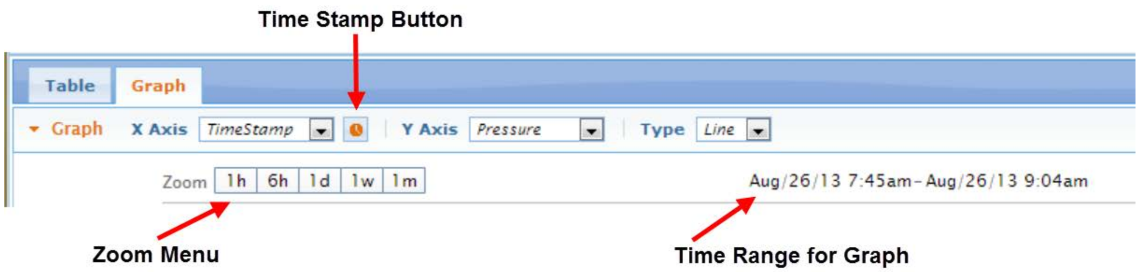
Click on the **Graph** tab to view the graphical data display for the selected vehicle and data type.

### Figure 8.6. Custom Reports Screen - Graph Format



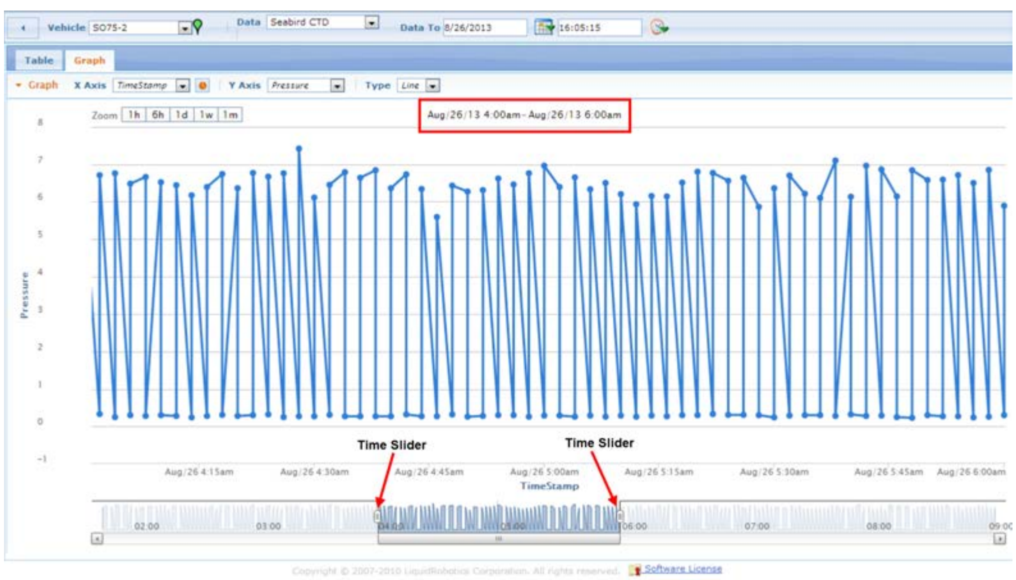
The selections made in the **X Axis** and **Y Axis** drop-down menus determine how the data is displayed. Displaying the **Time Stamp** on the X Axis with one of the other variables displayed on the Y Axis graphs the Y Axis data over time. This is the most widely used combination of X and Y Axis variables. The **Time Stamp** button adjacent to the **X Axis** drop-down menu can also be used to set the X Axis to **Time Stamp**. The **Type** drop-down menu is used to add or remove the line that connects the data points on the graph. Click on one of the selections in the **Zoom** menu to display data for the last hour, 6 hours, 1 day, 1 week, or 1 month. The Time Range indicates when the data display begins and ends.

### Figure 8.7. Custom Reports Screen - Graph Format Time Settings




The **Time Slider** buttons in the time line at the bottom of the **Graph** display can also be used to adjust the time range for the data display.

### Figure 8.8. Custom Reports Screen - Graph Format Time Slider




## Note

The options available in the **X Axis** and **Y Axis** drop-down menus may not always result in a useful or meaningful graph. Knowledge of the data being measured is necessary to choose X and Y Axis variables that make sense.

# Chapter 9. Troubleshooting

Troubleshooting the entire system from the C3 sensor to WGMS requires an understanding of the data path. This section describes the steps to:

- Confirm that the C3 sensor transmits raw data correctly when it is powered-on.
- Confirm that the PayloadInterface is receiving events.
- Confirm that the C&C and the payload are able to communicate so they can exchange commands and data.
- Confirm that the C3's telemetry packets are being transmitted from the C&C to WGMS.

## Power-On

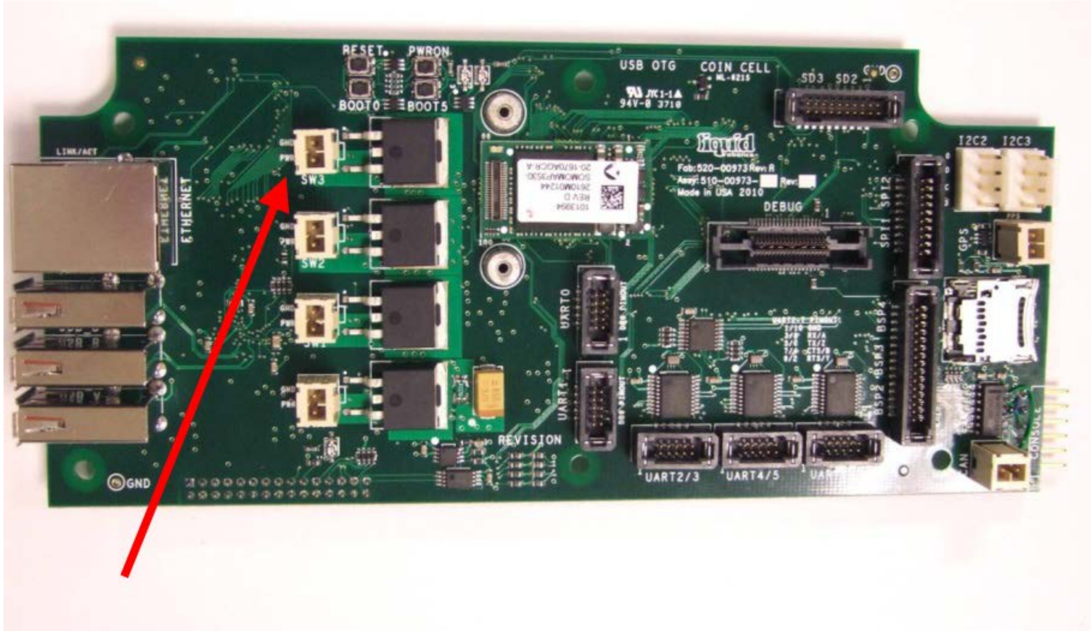
When the C3 sensor is turned on, it goes through the initialization sequence which is captured in the following log:

```
2013/01/02 21:55:55.778447 dm3730_torpedo [467] Entering OnProcessCommand
N14LiquidRobotics2C314C3PowerCommandE
2013/01/02 21:55:55.781775 dm3730_torpedo [467] Exiting
SendCommand command='C3 --on' requester='ServerExec-
6549-515608' transactionID='ServerExec-6549-515608-1': Caught
N14LiquidRobotics8EventLib26SensorControllerExceptions21ReturnsAsynchronouslyE(Command will return
asynchronously)
SetPowerSwitch: 0x00
2013/01/02 21:55:55.842047 dm3730_torpedo [467] Power OFF returned status 0
2013/01/02 21:55:57.842650 dm3730_torpedo [467] Set DTR
SetPowerSwitch: 0x88
2013/01/02 21:55:59.052758 dm3730_torpedo [467] Power ON returned status 0
2013/01/02 21:55:59.053979 dm3730_torpedo [467] Sleep 5 seconds
2013/01/02 21:56:04.056663 dm3730_torpedo [467] Clear DTR
2013/01/02 21:56:04.060113 dm3730_torpedo [467] Sleep 40 seconds
2013/01/02 21:56:51.482433 dm3730_torpedo [467] Firing CommandComplete event:
CommandComplete:SendCommandCompleteEvent/LiquidR-SMC requester='ServerExec-6549-515608'
transactionID='ServerExec-6549-515608-1' result=STRING/'<root create-time="
2013/01/02 21:55:55.778111" start-time="
2013/01/02 21:55:55.778600" end-time="2013/01/02 21:56:51.479013" cmd="C3 --on">
2013/01/02 21:56:51.485456 dm3730_torpedo [467] <output value="C3 ON"/>
2013/01/02 21:56:51.488112 dm3730_torpedo [467] </root>
2013/01/02 21:56:51.490341 dm3730_torpedo [467] ' complete=true
2013/01/02 21:56:51.518889 dm3730_torpedo [467] Exiting OnProcessCommand
N14LiquidRobotics2C314C3PowerCommandE: '<root create-time="2013/01/02 21:55:55.778111" start-time="2013/01/02
21:55:55.778600" end-time="2013/01/02 21:56:51.479013" cmd="C3 --on">
2013/01/02 21:56:51.529637 dm3730_torpedo [467] <output value="C3 ON" />
2013/01/02 21:56:51.532415 dm3730_torpedo [467] </root>
2013/01/02 21:56:51.541575 dm3730_torpedo [467] '
```

If you do not see this sequence in the `/home/smc/C3.log` file, there are 3 possible causes. The power line is not connected to the right power port, there are issues with the data line, or the C3 sensor is malfunctioning.

## Power Line

The C3 sensor power line should be connected to SW3 on the SMC.

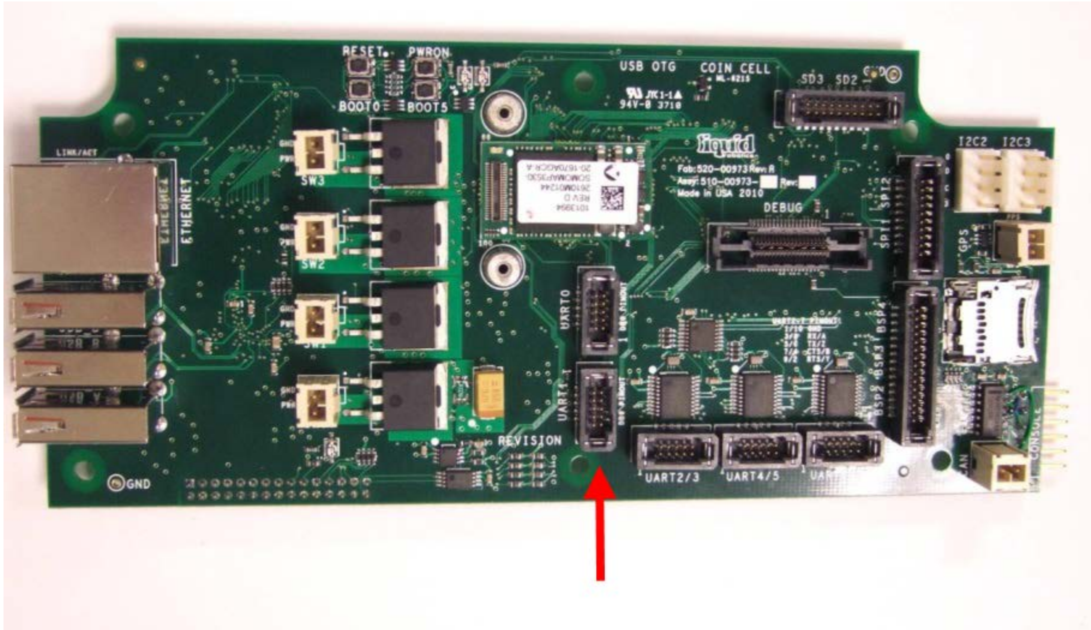



In the C3-config.xml file, SetPowerSwitch should be set to 4.

```
<powerOnVerb>sudo SetPowerSwitch 4 1</powerOnVerb>
<powerOffVerb>sudo SetPowerSwitch 4 0</powerOffVerb>
```

## Data Line

The C3 sensor data port should be connected to the UART1 port on the SMC.




The software configuration in the C3-config.xml file should be as follows:

```
<serialPortSettings class_id="15" tracking_level="0" version="0">
        <portName>/dev/ttyLX1</portName>
        <baudRate>57600</baudRate>
        <parity>NONE</parity>
        <characterSize>8</characterSize>
        <stopBits>1</stopBits>
        <flowControl>NONE</flowControl>
        <rts>1</rts>
        <dtr>1</dtr>
        <breakTimeoutMultiplier>1</breakTimeoutMultiplier>
</serialPortSettings>
```

If you are a developer and your hardware has never been used or tested by the LRI, consider swapping the TX/RX lines for debugging purposes.

## The C3 Sensor

If the power and data lines are connected and configured correctly, the only option left is to remove the sensor and use the C-Soft program provided by Turner Designs to verify the sensor.

# Data Validation

By default, C3 sensor data is written in the `c3raw.csv` file in the `/home/smc/roots/200_C3/` directory after you send the start command. Remember, it takes 60 seconds to power it on first. If data does not appear in the `c3raw.csv` file, either the wrong command was entered, the data samples are corrupted, or no data was received.

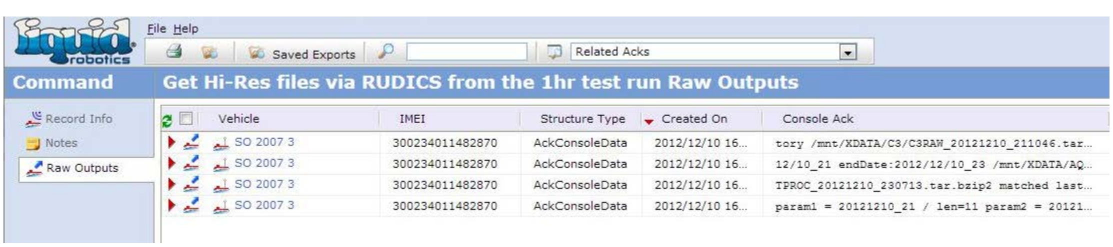
## Wrong Command

A command may have been entered incorrectly. The response always indicates the error, usually with a CORBA exception.

Example:

```
smc@dm3730_torpedo:~$ ServerExec C3 "C3 --config"  # should be --getconfig
Caught CORBA exception BadParameter with parameterName='command' and message='Unrecognized command
'C3 --config'' while executing command on C3
smc@dm3730_torpedo:~$ ServerExec C3 "C3 -start"  # should be --start
Caught CORBA exception BadParameter with parameterName='command' and message='Unrecognized command
'C3 -start'' while executing command on C3
```

You can also verify the command response in the Console ACK for Raw Outputs in WGMS.



## Corrupted Data

It is possible that the serial data line can produce errors under certain circumstances. You may be able to fix this problem by first changing `LT_INFO` to `LT_DEBUG` in the `C3-config.xml` file. After making this change, cycle the power to the payload - this may fix the corrupted serial line.

The change from `LT_INFO` to `LT_DEBUG` in the `C3-config.xml` file causes the C3 program to save the raw data in the `C3.log` file. If you do not see raw data in the `C3.log` file and you do not see processed data in the `c3raw.csv` file, the data may be corrupted. Try stopping and restarting the C3 sensor. If the problem persists, power-cycle the C3 sensor and then send a start command.

## No Data Received

If no data is written to the `c3raw.csv` file after you send a start command to the C3 sensor, you may not be waiting long enough. It takes 60 seconds for the C3 program to go through the initialization sequence. Wait for more than a minute and check the `c3raw.csv` file again.

You can also look for the C3 sensor initialization sequence in the `C3.log` file. If the initialization sequence appears in the `C3.log file`, the data is probably corrupted. In this case, change `LT_INFO` to `LT_DEBUG` in the `/home/smc/roots/200_C3/C3-config.xml` file and power cycle the payload. If you see the raw data coming in, but you don't see processed data in the `c3raw.csv` file, the data is probably corrupted, explaining why it was not written to the `c3raw.csv` file. Try stopping and restarting the C3 sensor stop again.

If you do not receive any data (raw or processed), go back and check the power status - see "Power-On" (page 42) for more information.

# Payload Interface and the Telemetry Packet

When the C3 sensor is working properly, data is written to the `c3raw.csv` file. `PayloadInterface` subscribes to C3 sensor events and forwards them to WGMS. If `PayloadInterface` processes the C3's events correctly, it will write telemetry events in the `PayloadInterface.log` as follows:

```
2013/01/02 18:49:31.462026 dm3730_torpedo [454] EventToTelemetry for
C3SampleDataAverage:C3SampleDataAverage/LqrSMAC
2013/01/02 18:49:31.483430 dm3730_torpedo [454] TelemetryReportComplete for publisherKey={0x2017}
formatCode=0x01 target 0x0002: 63 bytes { 91 3C 00 60 00 00 00 20 17 09 02 A5 81 56 01 37 04 A3
FB 83 80 E4 50 0D 00 1A 04 00 4D 07 00 2A 00 00 00 B5 08 A5 81 56 01 2F 04 A3 FB 3B 81 E4 50 0D
00 2F 02 00 15 07 00 0D 00 00 00 B7 08 }
```

At this point the data goes beyond the SMC to the C&C, following the data path to WGMS.

# WGMS not Receiving Data

If you are using an XBee modem, it may take 2 minutes before the C3 data samples are received by WGMS. It takes 1 minute for the C3 sensor to power-on, and the C3 program accumulates 7 data points before sending out a telemetry packet.

If you are using an Iridium modem, it can take 10 to 15 minutes before a C3 data sample is received by WGMS, due to Iridium transmission latency.

If you are using XBee, the first thing you should check is the communication between the payload and the C&C. The message `Polling Address: 2000` should show up in the vehicle XBee console. If you don't see this message, try setting the power delay for the payload as illustrated below, and then cycle power to the payload.


If you are using Iridium, the power-on delay has most likely been set by LRI. If you are not certain, send a power-on command to the payload to make sure it enumerates.

If WGMS still has not received data, go back to the beginning of this section and try to locate the problem. If the Wave Glider is at sea, you have no options other than power-cycling the C3 sensor and/or the payload. This is why it is important to verify that data is being received by WGMS <u>before</u> launching a mission.

# Stop Reporting When at Sea

There are a few common problems to consider if the C3 sensor suddenly stops transmitting data to WGMS:

- Iridium communications have a latency of several minutes and can occasionally cause errors.
- Another Wave Glider pilot might have stopped the sensor or powered down the payload without notifying you. You can confirm this by reviewing the command history in WGMS.
- If the payload runs out of power it will automatically shut down. Check the battery capacity or get an update from the piloting team.
- If none of the above circumstances exist, the C3 sensor may be having problems. Try cycling the power to the sensor and/or the payload.

# Unreasonable Data Values

If the data values are out of range, it is likely that the C3 sensor needs calibration. This is why it is important to calibrate the sensor before each mission.

# Appendix A. PIB and PM Slot Configuration Commands

## Overview

The LRI payload electronics architecture for Wave Glider SV2 vehicles includes a Payload Interface Board (PIB) with an 8 bit Atmel processor. Each PIB has three slots that can accommodate up to three generic device interface boards known as Personality Modules. A Personality Module (PM) provides a power and communications interface between a sensor (or other payload device) and the PIB. Each PM slot is configured to meet the power and data transmission requirements for the sensor or payload device it is connected to. For more information on PIB and PM electronics see the "Payload Development - Hardware Environment" chapter in the *Wave Glider SV2 Interface Control Description*.

On legacy Wave Glider SV2 vehicles, several LRI sensor-specific applications run on PIBs providing a management and communications interface between WGMS and sensors that LRI has integrated into the Wave Glider platform. On more recent Wave Glider SV2 vehicles, a single board computer known as a System Management Computer (SMC) is connected to a PM on a PIB with LRI sensor-specific applications running on the SMC Linux operating system.

This appendix provides information on the commands used to configure the PIB and its PM slots. This appendix does not provide information on LRI sensor-specific commands for managing LRI integrated sensor payloads.

If you have a legacy Wave Glider SV2 vehicle with an LRI integrated Turner C3 sensor, see Appendix B: *PIB C3 Sensor Program Commands* for information on the commands for the C3 application that runs on the PIB.

## Issuing PIB and PM Slot Configuration Commands

PIB and PM slot configuration commands are associated with several small programs that run on each PIB:

- PIBs are managed by the PIB program. The PIB program runs constantly when a PIB is powered on.
- The EMPTY program designates an unused PM slot as "empty." Issuing an EMPTY -i command to this PM slot will return a Console Data ACK containing the string `Dummy Sensor`. For more information see the section called “Empty ID”.
- The COMM program configures a PM slot to allow a connected device to communicate using the LRI payload communications protocol.
- The POWER program turns power to the PM slot on or off.
- The SMC program sets up a PM slot for a System Management Computer.
- The 37SI program is a sensor-specific application for managing the Seabird 37SI CTD sensor.

- The CTD program is a sensor-specific application for managing the Seabird GPCTD sensor.
- The C3 program is a sensor-specific application for managing the Turner C3 sensor.
- The ATM program is a sensor-specific application for managing the Teledyne ATM-88X Modem.
- The CO2 program is a sensor-specific application for managing the MBARI CO2 sensor.

PIB and PM slot configuration commands are sent from WGMS to the Wave Glider C&C using a Vehicle Command Form. All commands are sent to a combined Board Address and Task ID. The Board Address identifies a specific PIB and the Task ID is used to direct the command to the PIB, or to one of the PM slots on that PIB. Sensor-specific or payload device-specific commands must also be addressed to a PIB and the PM slot on that PIB where the sensor or payload device is connected.

To issue a command:

1. Select the **`Write Console Script`** command in the **Command Type** drop-down menu of the **Vehicle Command** tab.
2. Enter some text in the **Command Reason** field. This is a free-form field, but it is also a required field. If you enter nothing, you will not be able to issue the command.
3. Enter the decimal version of the combined Board Address/Task ID for a specific PIB or one of the PMs attached to it into the **Board Address** field.

   The combined Board Address/Task ID consists of two hexidecimal values. The first value is the Board ID for the PIB you want to issue a command to. The Board ID will be between `0x00` and `0xFF` - exclusive. For more information, see the section called “PIB ID”. The second value is the Task ID. The Task ID for a PIB is `00`. The Task IDs for each of the PM slots on a PIB are `01`, `02`, and `03`.

   For example:

   The combined Board ID/Task ID for a PIB with a Board ID of `0x01` would be `0x0100`. To determine the decimal version, multiply the Board ID value by 256 and add the Task ID {(01)(256) + 00}. The result is 256 . This is the number you would enter into the **Board Address** field on the **Vehicle Command** tab.

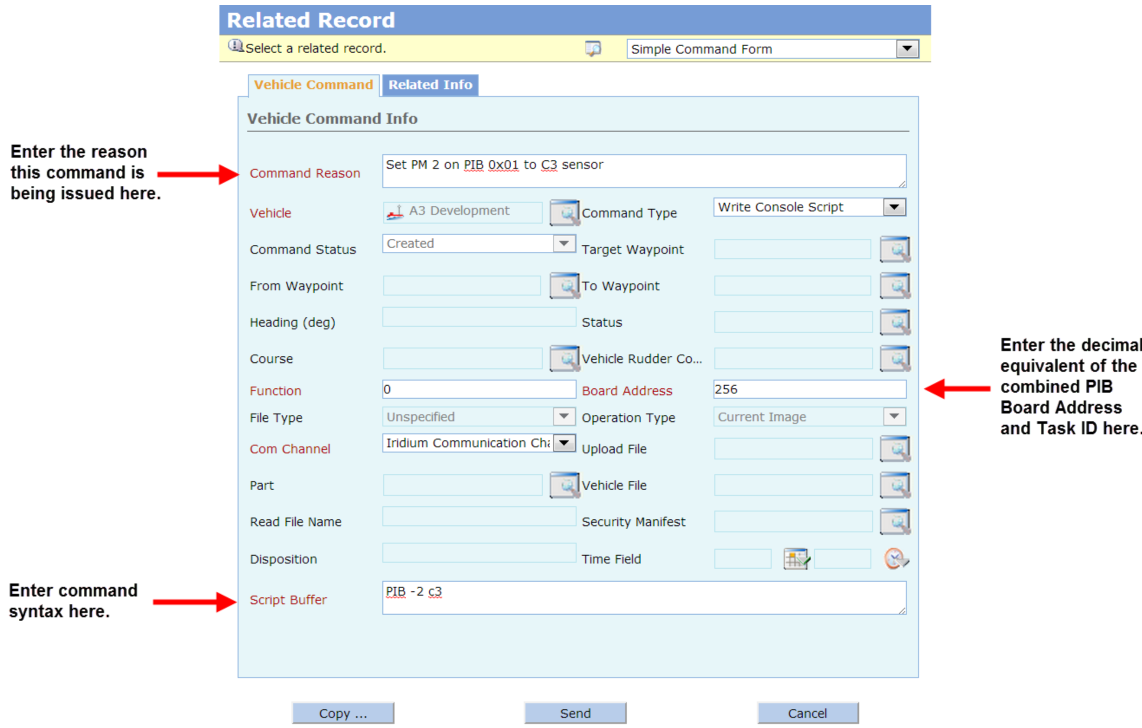
   If the Board ID for the PIB is `0x02`, and the command is directed to PM slot `03`, the combined hexidecimal values would be `0x0203`. To determine the decimal version, multiply the Board ID value by 256 and add the Task ID {(02)(256) + 03}. The result is 515. This is the number you would enter into the **Board Address** field on the **Vehicle Command** tab.
4. Enter the command you want to issue into the Script Buffer field in the Vehicle Command tab.
5. Click **Send** to issue the command.

### Note

PIB commands must be sent one at a time.

**All PIB commands are case-insensitive. Upper case characters are used in command syntax and examples for emphasis only.**

## Figure A.1. Vehical Command Form




# PIB PM Slot Configuration Commands

There are 3 commands used to configure each of up to 3 PM slots for the PMs that are connected to a PIB.

### Syntax

`PIB -1 | --pm1` [sensor]

`PIB -2 | --pm2` [sensor]

`PIB -3 | --pm3` [sensor]

### Summary

[sensor] assigns one of the PIB programs to the designated PM slot including the EMPTY program, if the PM slot is not being used.

### Semantics

These commands allow you to set the [sensor] configuration for each PM slot. The sensor parameter can be a string (case-insensitive) or a number.

| Program Setting | String | Number |
| --- | --- | --- |
| Empty Sensor | EMPTY | 0 |
| Payload Communications | COMM | 1 |
| System Management Computer | SMC | 2 |

| | | |
|---|---|---|
| Seabird 37SI CTD | 37SI | 3 |
| Seabird GPCTD | CTD | 4 |
| Turner C3 | C3 | 5 |
| Power On/Off | POWER | 6 |
| Teledyne ATM-88X Modem | ATM | 7 |
| MBARI CO2 | CO2 | 8 |

For example, sample commands are:

**PIB -3 ctd** This command tells the PIB to configure PM slot 3 with the Seabird GPCTD sensor-specific program.

**PIB --pm2 SMC** This command tells the PIB to configure PM slot 2 with the SMC program.

**PIB --pm1 0** This command tells the PIB to run the EMPTY program on PM slot 1. Note, the EMPTY program should be used for PMs that are not connected to anything. A PM configured with the Empty Sensor setting can only respond to an Empty ID command. For more information see the section called “Empty ID”.

Upon receiving a command for one of its PM slots, the PIB program will compare the current setting for that PM slot to the command argument (desired setting). If they are the same, no action will be taken. If they are different, the PIB program will clear the EEPROM allotted to that task and end the task. If the command is successful, the PIB program will send a Console Data ACK back to shore with the current PM slot configuration. Note that this configuration does not reflect what is currently running on the PIB, instead it reflects the program that will be loaded the next time the PIB is started. The desired program is stored in the PIB non-volatile memory and will run on the designated PM after a re-boot.

If the command is unsuccessful for any reason, the PIB will return a NAK. The most common cause of a NAK is an attempt to set an invalid sensor.

### Note

The PM configuration commands will NOT start the assigned programs, these commands are only for configuring the PM slots. Configuration setting changes will not take effect until the PIB is re-booted. The Power On/Off command must be used to start or stop the program assigned to a PM slot.

## Reading the PIB Configuration

**Syntax**

```
PIB -c | --cfg
```

**Summary**

Returns the current PM slot configuration.

**Semantics**

The PIB will return a Console Data ACK with the sensor configuration that is stored in the PIB nonvolatile memory. This is not necessarily the current setting - it is the setting that will run the next time the PIB is started. The response syntax is:

PM1=[*Sensor*] PM2=[*Sensor*] PM3=[*Sensor*]

For example, a response could be:

PM1=C3 PM2=SMC PM3=CTD

# PIB boardID

**Syntax**

`PIB -b | --boardID` [*boardID*]

**Summary**

Sets the board ID for the PIB.

**Semantics**

This command will set the board ID for the PIB. This ID must be between 0x00 and 0xFF, exclusive - otherwise the command will NAK. This board ID is stored in the PIB nonvolatile memory. The PIB will respond with a Console Data ACK containing an empty message.

## Note

This command will NOT take effect until the PIB is re-booted (power is cycled on the payload port). Commands sent to the PIB or any of its assigned sensors or functions must be addressed to the previous board ID, until power is cycled and the PIB re-boots at which time the new board ID will be set.

# PIB setDly

**Syntax**

`PIB -d | --setDly` [delayTime]

**Summary**

Sets the power down delay time to gracefully shut down the payloads (sensors or devices) connected to the PMs on the PIB. When the C&C receives a command to turn off the power to a payload port, the longest power down delay associated with a PIB connected to that port will determine the power down delay for the port.

**Semantics**

This command will set the power down delay time from 0 to 255 seconds. The power down delay time will be sent to the C&C the next time the PIB receives a power down notification. It is stored in the PIB nonvolatile memory.

# PIB ID

**Syntax**

`PIB -d | --id`

**Summary**

Returns PIB identification information.

**Semantics**

This command will respond with a Console Data ACK that contains identification information, including the board ID and the power down delay time. For example:

```
PIB ver. 2.0.341. Board ID = 0x01. Power Down Delay = 1 sec.
```

Note that the board ID will not necessarily be the current board ID; it will be the board ID setting that will be used the next time power to the PIB is cycled.

# PM Slot Commands

PM slot commands are a sub-set of PIB commands that designate unused PM slots, or configure power for PM slots connected to payloads that are not managed by an LRI sensor-specific application.

## Empty ID

**Syntax**

`EMPTY -i | --id`

**Summary**

Used to determine if there is nothing connected to a PM.

**Semantics**

If the `[sensor]` variable in any of the the section called "PIB PM Slot Configuration Commands" is set to `EMPTY` or `0`, the EMPTY id command will return a Console Data ACK containing the string `Dummy Sensor`. Any other command sent to this PM will return a NAK.

## Power On

**Syntax**

`POWER -o | --on`

**Summary**

Powers on the PM.

**Semantics**

This command will turn on the PM and then return a Console Data ACK indicating whether power is on or off.

## Power Off

**Syntax**

`POWER -f | --off`

**Summary**

Power off the personality module.

**Semantics**

This command will turn off the personality module. It will then return an Ack Console Data indicating whether power is on or off.

# Power AutoRun (mode)

**Syntax**

```
POWER -a | --autorun [mode]
```

**Summary**

Sets the Autorun mode.

**Semantics**

This command will turn the AutoRun feature On or Off. When AutoRun is On, the PM will be powered on when the PIB starts up. When Autorun is Off, the PM will not be powered on when the PIB starts. After the PIB is running, the power state can always be changed by sending a Power On or Power Off command. Valid parameters for the AutoRun command are `On, Yes, 1,` for Power On, or `Off, No, 0` for Power Off.

# Power Status

**Syntax**

```
POWER -s | --status
```

**Summary**

Sends back whether power is on or off.

**Semantics**

This command will return an Ack Console Data indicating whether power is on or off.

# Power ID

**Syntax**

```
POWER -i | --id
```

**Summary**

Sends back identification.

**Semantics**

This command will return an Ack Console Data containing identification.

# Appendix B. PIB C3 Sensor Program Commands

There are five commands for the C3 sensor program that runs on a PIB. These commands are used on Wave Glider SV2 vehicles with legacy PIB payload management hardware, as opposed to more recent SV2 vehicles that use an SMC for payload management.

### Note

**All C3 Sensor PIB commands are case-insensitive. Upper case characters are used in command syntax and examples for emphasis only.**

Commands for the PIB C3 program are sent from WGMS to the Wave Glider C&C using a Vehicle Command Form. All commands are sent to a combined Board Address and Task ID. The Board Address identifies a specific PIB and the Task ID is used to direct the command to a PM slot on that PIB that is connected to a C3 sensor.

To issue a command:

1. Select the **`Write Console Script`** command in the **Command Type** drop-down menu of the **Vehicle Command** tab.

2. Enter some text in the **Command Reason** field. This is a free-form field, but it is also a required field. If you enter nothing, you will not be able to issue the command.

3. Enter the decimal version of the combined Board Address/Task ID for a specific PIB or one of the PMs attached to it into the **Board Address** field.

   The combined Board Address/Task ID consists of two hexidecimal values. The first value is the Board ID for the PIB you want to issue a command to. The Board ID will be between `0x00` and `0xFF` - exclusive. For more information, see the section called “PIB ID”. The second value is the Task ID. The Task ID for a PIB is `00`. The Task IDs for each of the PM slots on a PIB are `01`, `02`, and `03`.

   For example:

   The combined Board ID/Task ID for a PIB with a Board ID of `0x01` would be `0x0100`. To determine the decimal version, multiply the Board ID value by 256 and add the Task ID {(01)(256) + 00}. The result is 256 . This is the number you would enter into the **Board Address** field on the **Vehicle Command** tab.

   If the Board ID for the PIB is `0x02`, and the command is directed to PM slot `03`, the combined hexidecimal values would be `0x0203`. To determine the decimal version, multiply the Board ID value by 256 and add the Task ID {(02)(256) + 03}. The result is 515. This is the number you would enter into the **Board Address** field on the **Vehicle Command** tab.

4. Enter the command you want to issue into the Script Buffer field in the Vehicle Command tab.

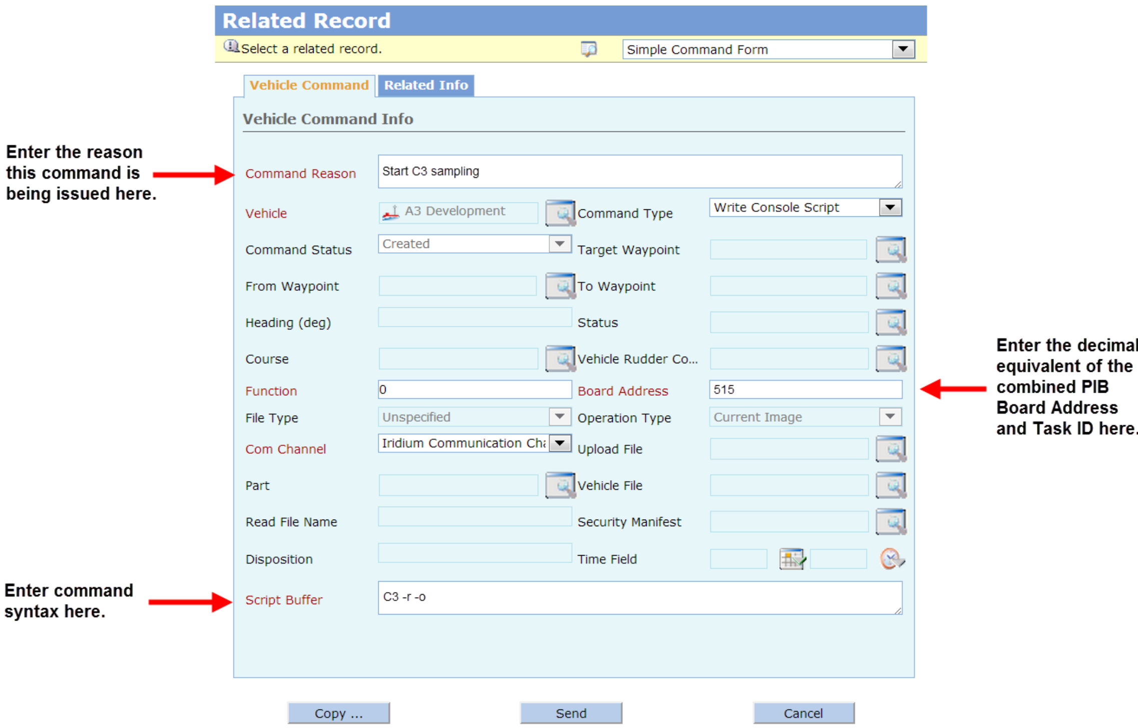
5. Click **Send** to issue the command.

## Note

Commands must be sent one at a time.

**All C3 commands are case-insensitive. Upper case characters are used in command syntax and examples for emphasis only.**

### Figure B.1. Vehical Command Form




# C3 Run / C3 On

**Syntax**

```
C3 -r | --run | -o | --on
```

**Summary**

Starts C3 sampling.

**Semantics**

This command will simultaneously power on the C3 and start C3 sampling. A Console Data ACK containing the message "OK" will be returned if the command is successful.

# C3 Stop / C3 Off

**Syntax**

```
C3 -x | --stop | -f | --off
```

**Summary**

Stops C3 sampling.

**Semantics**

This command will simultaneously stop C3 sampling and disable power. Any remaining data reports will be sent back to WGMS. A Console Data ACK containing the message "`OK`" will be returned if the command is successful.

# C3 AutoRun [mode]

**Syntax**

`POWER -a | --autorun` [mode]

**Summary**

Sets the C3 AutoRun mode.

**Semantics**

This command will turn the C3 AutoRun feature On or Off. When AutoRun is On, the PM will be powered on when the PIB starts up. When AutoRun is Off, the PM will not be powered on when the PIB starts. After the PIB is running, the C3 power state can be changed by sending a `C3 Run/C3 On` or `C3 Stop/C3 Off` command. Valid parameters for the AutoRun command are `On, Yes, 1,` for AutoRun On, or `Off, No, 0` for AutoRun Off.

# C3 ID

**Syntax**

`C3 -i | --id`

**Summary**

Sends back the C3 identification.

**Semantics**

A Console Data ACK containing identification information will be returned if the command is successful.

# C3 Status

**Syntax**

`C3 -s | --status`

**Summary**

Sends back the C3 status.

**Semantics**

A Console Data ACK with the current status of the C3 and confirmation that the C3 is sampling will be returned if the command is successful.

# Glossary

| | |
|---|---|
| C3 | The Turner C3 Submersible Fluorometer. |
| `C3` | The prefix for issuing commands to, and displaying responses from, the C3 sensor (case insensitive).<br><br>Example: `ServerExec C3 “C3 --on"` |
| C3 | The configurable sensor’s name to allow the Event Manager to recognize the C3 program (case sensitive).<br><br>Example: ServerExec C3 “C3 --on” |
| Script | A sequence of C3 commands collected in a file and executed in sequential order by the C3 process. |
| C&C | The Wave Glider Command and Control module. |
| LRI | Liquid Robotics, Incorporated |
| SMC | Sensor Management Computer – an embedded Linux application processor. |
| WGMS | Wave Glider Management System – the LRI shore-side control application. |