Liquid Robotics Proprietary and Confidential

Wave Glider® SV3 Interface Control Description

Communications and Control Interfaces for Payload Integration

Revision C
April 2016
Part Number 04921

LIQUID ROBOTICS®

Wave Glider® SV3 Interface Control Description: Communications and Control Interfaces for Payload Integration

Copyright © 2016 Liquid Robotics, Inc.

Confidential computer software. Valid license from Liquid Robotics, Inc. required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Trademarks

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

Patents

The Wave Glider mechanism is patented under US Patent Number 7,371,136 and other patents pending. All aspects of the Wave Glider product have been developed at private expense by Liquid Robotics, Inc.

Warranty

The information contained herein is subject to change without notice. The only warranties for Liquid Robotics, 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. Liquid Robotics, Inc. is not liable for technical or editorial errors or omissions contained herein.

Revision History

Document Part Number	Revision	Publication Date	Changes
04921	A	December 2013	This is the initial release of this document.
04921	B	September 2014	Minor changes to Chapter 2, "Mechanical Specifications." Updates to Chapters 5, 6, 7, and 8 reflecting changes to commands, responses, and telemetry. Added 1-MPU Payload Development Kit to Chapter 9 "Payload Development Mechanical and Electrical Considerations." New information on AMPS modules in Chapter 10, Payload Development Hardware Environment." Extensive new content in Chapter 11, "Payload Development Software Environment."
04921	C	April 2016	Updates to: Chapter 2, "Mechanical Specifications", Chapter 3 "Electrical Specifications", Chapter 4 "CCU Remote Communications and Control", Chapter 7 "Vehicle Telemetry", Chapter 9 "Payload Development Mechanical and Electrical Considerations", Chapter 11 "Payload Development - Software Environment." Also a new Appendix "Towfish Battery Charger Specifications" was added.

Table of Contents

About this Document xv
1. Overview 1
Wave Glider Major Components 1
Payload Integration 3
2. Mechanical Specifications 4
Float Assembly 4
Recovery Buoy and Release Mechanism 6
Float Overall Dimensions 8
Command and Control Unit (CCU) 8
CCU Weight and Dimensions 9
Float Payload Bays 9
Solar Panels 10
Solar Panel Specifications 11
Shading 11
Float to Sub Umbilical 12
Sub Recovery Mechanism 16
Motion Isolator 17
Sub 20
Sub Dimensions and Payload Attachment Points 23
Thrudder Module 24
Towfish Options 26
Gateway Towfish 26
PAM Buoy Towfish 27
Tow Cable 28
Custom Towfish 29
3. Electrical Specifications 31
Physical Connectivity 31
CCU Electrical 32
G1, G2, G3 - General Purpose Expansion Ports 34
S1, S2, S3, S4 - Sensor Connections 34
HV - High Voltage Connection 34
U1 - Umbilical 34
D1 - Debug Port 34
O1 - System Connection 34
O2 - System Connection 34
Power Key 34
Power Key Indicator Light 35
Payload Connectivity 35
Payload Communication Connectivity 35
Payload Power Connectivity 36
GPS and PPS Connectivity 37
Adaptable Modular Power System (AMPS) 38
CCU Standard Electrical Connections 41
CCU Expansion Ports 41
CCU Sensor Ports 47
CCU High Voltage Connector 49
CCU System Connections 50
Thrudder Module Electrical Connections 61
Overview 61
Thrudder Module Umbilical Connector 63
Thrudder Module Expansion Port Connector 65

Liquid Robotics Proprietary and Confidential

Unused Connections 66
Batteries 66
Battery System Power 66
Battery Charge Level 66
Battery Discharge Characteristics 67
Battery Power Input 67
External Battery Charging 68
Liquid Robotics Multi-Function Debug/Charging Cable 68
Liquid Robotics Battery Charger 68
Charging the Batteries 69
APU Electrical Connections 70
A1, A2 - APU Expansion Port Connectors 73
4. CCU Remote Communications and Control 75
CCU Communications Functions 75
Telemetry and Polling 75
Command Processing 76
Command and Message Responses 76
CCU Communications Channels 76
Iridium Relay Link 77
CCU Communications Security 78
CCU Communications Components 78
Communication Relays 79
Standard Modems 80
Iridium Network Servers and Satellites 80
CCU Communication Data Types 80
Telemetry 81
Commands 81
Responses 81
Communication Packets 81
Relay Data Packets 83
Addresses 84
Vehicle Unique ID 84
Transaction ID 84
Communication Data Formats 84
Little-Endian Byte Format 84
Waypoint Values 85
Heading Values 85
Distance Values 85
Time Values 85
Heading Values 85
Distance Values 85
Temperature Values 86
File Names 86
GPS 86
Navigation 87
Navigation Modes 87
5. Commands 89
Command Data Packet Format 89
Command Summary 90
Commands for SV3 Vehicles Only 90
SV2 Commands for SV3 Vehicles 90
6. Vehicle Responses 92
ACK/NAK Protocol 92
ACK/NAK Packet Format 92

Liquid Robotics Proprietary and Confidential

ACK Data Type Numbers 93
NAK Data Type Numbers 93
Unsolicited Response Type Numbers 93
Relay Messages 94
ACK Response Data 94
ACK - Status with Waypoint Read-Back (0X10) 94
ACK - Status Return (0x11) 95
ACK - Vehicle Identification Data (0x12) 96
ACK - No Payload Response (0x13) 99
ACK - Power Controller Status (0x14) 99
ACK - Read File Block (0x1B) 100
ACK – Console Data (0x3D) 101
NAK Response Data 101
NAK - Invalid Command (0x22) 101
NAK – Invalid Data (0x22) 101
NAK – File Error (0x24) 101
NAK – Security Failure (0x25) 102
NAK – Console Data (0x26) 102
NAK – With Description (0x27) 103
Unsolicited Messages 103
New File Available (0x60) 103
Send Console Output to Client (0x1D) 104
Relay-Only Message 104
Relay Status (0x70) 104
Vehicle File System Responses 105
New File Available 105
Read File Block ACK 106
7. Vehicle Telemetry 107
Telemetry Packet Format 107
Telemetry Types 107
Telemetry Buffering 108
Encrypted Output Packet 0xA6 108
Compressed (GZIP) Output Packet 109
Compressed (BZIP2) Output Packet 109
Multi-L2 Output Packet 109
Computed Course Telemetry Packet Type 13 110
Resilient Tracker Telemetry 111
NAL Tracker Telemetry 0x03 111
Shout Tracker Telemetry 0x05 112
Standard Telemetry Data 0x06 114
Alarm Bytes 116
Float Status Byte 116
Telemetry - Power Controller Status 0x7 117
DB3.5 Power Control Status 0x08 117
Twist Status 0x09 121
Mobile WGMS Heartbeat 0x10 122
Mobile WGMS Sync Packet 0x11 122
Relay Status 0x70 122
System Information 0x71 123
System Information Records 124
System Information Types 124
Partition State Record (Type 1) 124
Single Payload Packet 0x80 125
Weather Station Packet (Type 1) 125

Liquid Robotics Proprietary and Confidential

Payload Status Packet (Type 2) 126
Payload Enhanced Pressure Weather Station Report Type 5 127
Payload AIS Status Packet (Type 16) 127
Payload AIS Static Record (Type 17) 128
Payload AIS Position Report (Type 18) 129
Sensor Sample Record (Unparsed) 129
Fluorometer (C3) Data Type 96 130
Compressed CTD Data Type 112 130
Compressed CTD Data 130
Compressed CTD With Oxygen Sensor Data 131
Payload ECO Puck Sensor Data Type 123 131
Payload CStar Sensor Data Type 124 133
Payload Benthos Modem Data Type 125 134
Payload Datawell MOSE Sensor Data Type 126 134
Payload VEMCO VR2C Data Type 127 135
Payload ADCP Data Type 128 135
Payload Aquatracka Data Type 130 136
Payload WHOI Micro Modem Type 131 137
Harmony Payload Type 132 138
Cefas OrbComm Payload Type 134 138
Turbidity Payload Type 135 139
Sonardyne Tsunami Sensor Payload Type 138 139
Marine Magnetics Magnetometer Type 140 140
Payload CO2 Sensor Type 144 141
Sentinel Sensor Data Type 146 142
GPS Wave Sensor Data Type 147 143
CSIR Sensor Data Type 148 144
MAGI Sensor Type 150 144
Payload Thermistor Array Data Type 192 145
Payload ASTRA GPS Data Type 259 146
Multi-Payload Packet 0x81 146
Payload Packet 0x91 146
Sensor Sample Records 147
8. CCU Payload Communications 148
Sensor Port Payload Communications Protocol 148
Communication Method 148
Protocol Message Format 151
Addresses 153
Other Considerations 155
Payload Message Types 155
Payload Message Packet Descriptions 156
Message Protocol Use Cases 167
9. Payload Development Mechanical and Electrical Considerations 172
Float Payload Considerations 172
Weight 172
Center-of-Gravity 173
Sail Area Considerations 173
Deck Mounting Considerations 174
Towing Considerations 175
Payload Power Considerations 175
Power Requirements - Evaluating Payload Integration Feasibility 175
Power Requirements - Evaluating Wave Glider Mission Feasibility 176
Sustainable Payload Power 176
Adapting Wave Glider SV2 Payloads for use on the Wave Glider SV3 177

Liquid Robotics Proprietary and Confidential

Mechanical Considerations 177
Electrical Considerations 179
Modular Payload Dry Boxes 180
Weight of Modular Payload Dry Boxes 182
1-MPU Payload Dry Box 182
3-MPU Payload Dry Box 184
Auxiliary Power Unit (APU) 185
1-MPU Payload Development Kit 186
3-MPU Payload Development Kit 189
Sub Payload Considerations 191
Drag 191
Sub Ballast 192
Wing-Payload Interference 193
Balance and Flotation 194
Poor Sail Area Locations 195
Acceptable Sail Area Locations 195
Trailing Wires 195
Towfish Payload Considerations 197
10. Payload Development - Hardware Environment 198
AMPS Modules 198
Power Domain Controller (PDC) Module 198
Battery Charger Module 201
Solar Panel Input Module 202
High Voltage Module 203
Buck-Boost Regulated Output Module (BBR) 206
Bridge Module 208
1 MPU Module Carrier 209
3 MPU Module Carrier 211
Thrudder PDC 213
SMC Carrier 215
Sensor Management Computer (SMC) 217
Overview 218
Console Port 218
Full RS-232 Ports (UART0 and UART1) 219
Configurable UART Ports (UART2/3, UART4/5, and UART6/7) 220
GPS/PPS Port 220
Power Ports 221
CAN Port 222
USB Ports (not USB OTG) 222
USB OTG Port 222
Ethernet Port 223
SD Card Expansion Port 223
SPI Expansion Port 224
McBSP Expansion Port 225
I2C Ports 226
11. Payload Development - Software Environment 228
Regulus 228
SV3 Manifests 228
How Remote Applications Handle Manifest Files 229
Parsing the Manifests 230
Expansion Port Payload Registration with Regulus 239
Regulus Expansion Port REST Interface (XPREST) 243
Technologies 243
Functions 250

Liquid Robotics Proprietary and Confidential

Additional Information on Payload Software Development 255
12. Communication Channels for Payload Development and Testing 257
13. Troubleshooting 258
Overview 258
Iridium Interface 258
General Checks 258
Re-Booting 258
Contacting Liquid Robotics 259
Problems and Responses 259
No Iridium Connection 259
No GPS Reception. 259
Power Key Light Does Not Blink 259
No Float Payload Power From CCU 259
No Sub Payload or Towfish Payload Power From CCU 260
No CCU-to-Payload RS-232 Communications 260
Float Continuously Re-Boots 260
Sub Continuously Re-Boots 260
Interpreting CCU Alarms 260
Sub Leak Alarm (Byte 0 | bit 0) 261
Float Leak Alarm (Byte 0 | bit 1) 261
Sub-to-Float Comms Alarm (Byte 0 | bit 2) 262
Payload Alarm (Byte 0 | bit 3) 262
Float Battery Low Alarm (Byte 0 | bit 4) 262
Umbilical Fault Alarm (Byte 0 | bit 5) 262
GPS Not Functioning (Byte 0 | bit 7) 262
Sub Reboot Alarm (Byte 1 | bit 1) 263
Float-to-Sub Comms Error (Byte 1 | bit 2) 263
Float Temperature Threshold Exceeded (Byte 1 | bit 4) 263
Float Pressure Threshold Exceeded (Byte 1 | bit 5) 263
EPROM Error (Byte 1 | bit 6) 263
Float Reboot Alarm (Byte 1 | bit 7) 264
Contacting Liquid Robotics, Inc. 264
A. Command Descriptions 265
Commands for SV3 Vehicles Only 265
SV3-Only Command Syntax 265
SV2 Commands for SV3 Vehicles 268
Hold Current Position (0x0001) 268
Hold Station at Waypoint (0x0002) 268
Follow Sequential Waypoint Course (0x0003) 269
Rudder Right/Left/Center (0x0004, 0x0005, 0x0006) 269
Follow Fixed Heading (0x0008) 270
Set Rudder Position (0x1006) 270
Visible Light (Marker Light) Off/On (0x2001, 0x2002) 270
Retrieve Status (0x4001) 271
Set Status (0x4002) 271
Retrieve Identification (0x4003) 272
Retrieve Power Status (0x5001) 273
New Waypoint (0x8000) 273
Read Waypoint (0x8003) 273
Follow Course and Loop (0x8101) 274
Follow Course With Options (0x8102) 275
Read File Request (0x9003) 275
Read File Transaction Closed (0x9007) 276
Write Console Script (0x9100) 277

Liquid Robotics Proprietary and Confidential

Float Reboot (0xA000) 277
B. Configuration Manifest Example 278
C. Commands Manifest Example 296
D. Towfish Battery Charger Technical Specifications 319
Electrical Specifications 320
Connectors and Pinouts 321
Functional Description 322
Calculating Power and Load Requirements 324
E. Relay Header Data 326
Iridium Vehicle-to-Remote Applications Data Packets 326
Mobile Originated Lat/Long Data 327
Relay Vehicle-to-Remote Applications Packets 327
Virtual Relay Vehicle-to-Remote Applications Packets 329
Iridium Remote Applications-to-Vehicle Data Packets 330
Relay Remote Applications-to-Vehicle Packets 331
Virtual Relay Remote Applications-to-Vehicle Packets 332
F. CRC-16 Calculation 333
G. PIB and PM Slot Configuration Commands for SV2 Payloads Adapted to SV3 Vehicles 334
Overview 334
Issuing PIB and PM Slot Configuration Commands 334
PIB PM Slot Configuration Commands 336
Reading the PIB Configuration 337
PIB boardID 338
PIB setDly 338
PIB ID 338
PM Slot Commands 339
Empty ID 339
Power On 339
Power Off 339
Power AutoRun (mode) 340
Power Status 340
Power ID 340

List of Figures

1.1. Wave Glider System (Float, Umbilical, and Sub) 2
1.2. Optional Towfish 2
2.1. Float Assembly Exploded View 4
2.2. Float Integral Features 5
2.3. Float Overall Dimensions (mm) 8
2.4. CCU 8
2.5. CCU Dimensions (mm) 9
2.6. Float Payload Bay Dimensions [inches] 10
2.7. Standard Solar Panels 10
2.8. Umbilical Assembly Overview 13
2.9. Umbilical-to-Float Connection 14
2.10. Umbilical-to-Sub Connection 15
2.11. Sub Recovery Mechanism 16
2.12. Sub Recovery Mechanism Locked Onto Recovery Hooks 16
2.13. Motion Isolator - Top View 17
2.14. Motion Isolator Extended, Cut-Away View 18
2.15. Motion Isolator Compressed, Cut-Away View 19
2.16. Sub 20
2.17. Sub with Thrudder Turned - Side View 21
2.18. Sub with Thrudder Turned - Top View 21
2.19. Sub Payload With Custom Fairing 22
2.20. Sub Payload With Custom Fairing - Translucent 22
2.21. Sub Overall Dimensions - mm [inch] 23
2.22. Sub Attachment Points mm [inch] 23
2.23. Thrudder Module 24
2.24. Thrudder Attachment Exploded View 25
2.25. Thrudder Attached to Sub with Fairings Installed 25
2.26. Gateway Towfish 26
2.27. PAM Buoy Towfish 27
2.28. Towfish with Tow Cable 3D 28
2.29. Towfish with Tow Cable Side View 28
2.30. Example of a Modular Towfish 29
2.31. Example of a Large Payload Towfish - Dimensions in Inches 30
3.1. Wave Glider CCU Connection Diagram 31
3.2. CCU Electrical Connections 33
3.3. CCU Electrical Block Diagram 33
3.4. Payload Communications Diagram 36
3.5. Payload Power Diagram 37
3.6. GPS and PPS Diagram 38
3.7. AMPS Power Bus 40
3.8. O1 Octopus Cable 51
3.9. O2 Octopus Cable 52
3.10. Octopus Cable Assemblies Mounted to Bulkhead 53
3.11. Thrudder Module Electrical Connections 61
3.12. Thrudder Module Electrical Block Diagram 62
3.13. Battery Charge Level Relative to Voltage 67
3.14. Battery Discharge Time vs. Current Draw 67
3.15. Liquid Robotics Multi-Function Debug/Charging Cable 68
3.16. Liquid Robotics Battery Charger 69
3.17. Auxiliary Power Unit (APU) Electrical Connections 70
3.18. APU Internal Battery Pack 71

Liquid Robotics Proprietary and Confidential

3.19. APU Electrical Diagram 71
4.1. Wave Glider Remote Communications 75
4.2. Vehicle to Remote Applications Iridium Communications 77
4.3. CCU Communications Components 79
4.4. Level 3 Relay Data Packet Format 82
4.5. Level 3 Encrypted Relay Data Packet Format 82
4.6. Level 2 Payload Data Packet Format 83
4.7. Level 3, Level 2, and Level 1 Data Packets 83
4.8. GPS Satellite Constellation 86
5.1. Command Packet Format 89
6.1. ACK/NAK Response Packet Format 92
7.1. Level 2 Telemetry Data Packet 107
8.1. Payload Data Packet Format 150
8.2. Message Packet Format 152
9.1. Float Max Forward Center-of-Gravity 173
9.2. Float Max Aft Center-of-Gravity 173
9.3. Possible Sail Area Problems 174
9.4. Deck Mounted Payloads 174
9.5. Towing a Payload Behind Float 175
9.6. Solar Power Relative To Time and Location 176
9.7. SV2 Forward Payload Dry Box - SV3 Mounting Kit (Liquid Robotics Part Number 04731) 178
9.8. SV2 Aft Payload Dry Box - SV3 Mounting Kit (Liquid Robotics Part Number 04732) 178
9.9. SV2 Short Aft Payload Dry Box - SV3 Mounting Kit (Liquid Robotics Part Number 05224) 179
9.10. 3-MPU Assembly 181
9.11. 1-MPU 182
9.12. 1-MPU Interior Dimensions (mm) 183
9.13. 1-MPU Attachment Points (mm) 183
9.14. 3-MPU 184
9.15. 3-MPU Interior Dimensions (mm) 185
9.16. 3-MPU Attachment Points (mm) 185
9.17. Auxiliary Power Unit (APU) Dry Box 186
9.18. 1-MPU Payload Development Kit Connectors 187
9.19. 1-MPU Payload Development Kit Electronics A 188
9.20. 1-MPU Payload Development Kit Electronics B 188
9.21. 1-MPU Payload Development Kit Electronics Block Diagram 189
9.22. 3-MPU Payload Development Kit Connectors 190
9.23. 3-MPU Payload Development Kit Electronics 190
9.24. 3-MPU Payload Development Kit Electronics Block Diagram 191
9.25. Sub Moving Up 191
9.26. Sub Moving Down 192
9.27. Poor Payload Design/Location 192
9.28. Sub Ballast 193
9.29. Sub Nose Assembly with Ballast 193
9.30. Wing-Payload Interference Area 194
9.31. Checking Sub Underwater Balance 194
9.32. Excessive Forward Payload Sail Area 195
9.33. Excessive Aft Payload Sail Area 195
9.34. Acceptable Payload Sail Area Location 195
9.35. Poor Trailing Wire Location (Tow Point) 196
9.36. Poor Trailing Wire Location (Tow Point) 196
9.37. Acceptable Trailing Wire Location (Tow Point) 196
10.1. AMPS Power Domain Controller Module (PDC) 199
10.2. PDC Electrical Diagram 201
10.3. Battery Charger Module 202

Liquid Robotics Proprietary and Confidential

10.4. Battery Charger Module Electrical Diagram 202
10.5. Solar Panel Input Module 203
10.6. Solar Panel Input Module Electrical Diagram 203
10.7. High Voltage Module 205
10.8. AMPS Buck-Boost Regulated Output Module (BBR) 206
10.9. Buck-Boost Power Diagram 208
10.10. Bridge Module 209
10.11. Bridge Module Electrical Diagram 209
10.12. 1 MPU Module Carrier 210
10.13. 3 MPU Module Carrier 212
10.14. AMPS Thrudder Power Domain Controller Module (PDCT) 214
10.15. Thrudder Power Domain Controller Electrical Diagram 215
10.16. SMC Carrier Board with SMC Mounted 216
10.17. SMC Connector Locations 218
10.18. Console Port Connection 219
10.19. GPS/PPS Connections to a PIB 221
11.1. Manifest Files as WGMS Attachments 230
11.2. Parsing Flow 238
11.3. SV3 Payload Registration and Manifest Publication 239
D.1. Towfish Charger Block Diagram 320
D.2. Functional Flow Chart 322
G.1. Vehical Command Form 336

List of Tables

2.1. SV3 Solar Panel Specifications 11
2.2. Solar Panel Shading Considerations 11
3.1. Color and Label Conventions for Wave Glider Connection Diagram 31
3.2. Expansion Port Signals 42
3.3. Pins 1 and 8 48
3.4. Pins 2 and 7 49
3.5. Pins 3, 4, 5, and 6 49
3.6. Thrudder Module Umbilical Connector and Expansion Port Signals 62
3.7. Liquid Robotics Battery Charger Specifications 69
3.8. APU Expansion Port Signals 72
4.1. Navigation Modes 87
5.1. Command Summary 90
7.1. Telemetry Types 108
9.1. Weight of Float and Standard Accessories 172
9.2. Maximum Payload Weight 172
9.3. Weight of Modular Payload Dry Boxes 182
10.1. System On/Off (J10) 199
10.2. Leak Detect Port Pin Input (J8) - currently not used 199
10.3. CAN Bus (J9) 200
10.4. Console (J4) 200
10.5. Electrical Specifications 201
10.6. High Voltage Connector Pin Out (4-pin) 206
10.7. Switched Vconsumer Pin Out (2-pin) 206
10.8. Electrical Specifications 207
10.9. Port Status LED Warning Conditions 207
10.10. Output Power Ports (2 Pin, J5 - J11) 207
10.11. Electrical Specifications 210
10.12. Expansion Port Signals 210
10.13. Expansion Port CAN IN 211
10.14. Expansion Port Power In 211
10.15. Spare Lines Breakout 211
10.16. Electrical Specifications 212
10.17. J2 - Expansion Port Signals 212
10.18. J21 - Expansion Port CAN IN 213
10.19. J1 - Expansion Port Power In 213
10.20. J4 - Spare Lines Breakout 213
10.21. J22 - Power Breakout 213
10.22. Console (J9) 214
10.23. Electrical Specifications 216
10.24. J3 - Sensor Port Landing 217
10.25. J5 - GPS/PPS Breakout 217
10.26. J11 - RS232 Breakout 217
10.27. J10 - Power Breakout 217
11.1. Configuration Manifest Parameters 234
11.2. Command Manifest Parameters 236
13.1. CCU Alarms 261
A.1. Write Console Script Binary Command Structure 265
D.1. Absolute Maximum Ratings 320
D.2. Recommended Operating Conditions 321
D.3. Connector List 321
D.4. Battery Interface Connector Pinout 321

Liquid Robotics Proprietary and Confidential

D.5. DIP Switch Settings 322
D.6. State of Charge Thresholds 322
D.7. LED Interface Indicators 323

About this Document

This document is a reference to Wave Glider SV3 mechanical, electrical, communications, and software interfaces commonly encountered in payload development projects.

- Chapters 1-8 define the interfaces for payload development.
- Chapter 9 provides guidance on some key payload development parameters.
- Chapters 10 and 11 contain information on Liquid Robotics payload development hardware and software designed to streamline the payload development process.
- Chapter 12 discusses communications channels available for payload development and testing.

Note

This document does not contain information on the Wave Glider Management System (WGMS) - a Liquid Robotics remote Wave Glider navigation and communications application. For information on WGMS, see the *WGMS™ User Guide* and the *WGMS™ Administrators Guide*.

The term "remote application" is used in this document to refer to all remote Wave Glider navigation and communications applications, including WGMS.

Important

The term Payload Interface Board or PIB is mentioned several times in this document. PIBs are part of the Wave Glider SV2 payload development hardware environment, but they can also be included in a Wave Glider SV3 configuration when an SV2 payload is adapted for service on an SV3. For more information on PIBs see the Wave Glider SV2 Interface Control Description and also see the following sections in this document:

- “Adapting Wave Glider SV2 Payloads for use on the Wave Glider SV3” (page 177)
- “S1, S2, S3, S4 - Sensor Connectors” (page 47)
- Appendix G: *PIB and PM Slot Configuration Commands for SV2 Payloads Adapted to SV3 Vehicles*

Chapter 1. Overview

The Wave Glider®® is an autonomous ocean-going platform for deploying custom ocean-to-surface sensor systems. This chapter describes the major components of the Wave Glider SV3 system and the key parameters impacting payload integration feasibility.

Wave Glider Major Components

The Wave Glider system has seven major components:

- **Float** —The Float is the component that rides on the surface of the water. It contains the batteries, control and communications electronics, and optional payloads. Three solar panels on the Float deck maintain the battery charge. The Float deck typically has additional sensing and communications devices installed. A buoy with a retrieval line can be released from the back of the float to simplify recovery. For more information see:
  - "Float Assembly" (page 4)
- **Umbilical** —The Umbilical is a high-strength streamlined connector between the Float and the Sub. The Umbilical provides electrical power to the Sub and also contains wiring for data communications between the Float and Sub. For more information see:
  - "Float to Sub Umbilical" (page 12)
- **Sub** —The Sub is always submerged well below the Float during operation. It provides the propulsion and steering for the Wave Glider system. The Sub can also be used as a platform to mount underwater sensing equipment or to attach a Towfish. Power and communications wiring connections on the Sub Thrudder module can accommodate these sub-surface payloads. For more information see:
  - "Sub" (page 20)
- **Towfish (optional)** —Towfish are streamlined, buoyancy neutral, sub-sea extensions of the Wave Glider system that provide mounting points, power and communications channels for a wide variety of underwater payloads. For more information see:
  - "Towfish Options" (page 26)
- **Electrical System** —Command and Control Unit (CCU) including electrical and payload communications hardware, standard power and communications connections, solar panels and batteries. For more information see:
  - Chapter 3: *Electrical Specifications*
  - Chapter 10: *Payload Development - Hardware Environment*
- **Communications** —Communication methods, commands, protocols, and relays. For more information see:
  - Chapter 4: *CCU Remote Communications and Control*
  - Chapter 5: *Commands*
  - Chapter 6: *Vehicle Responses*

Liquid Robotics Proprietary and Confidential

– Chapter 7: *Vehicle Telemetry*

- **Software** —Developing payload-specific applications that run on a System Management Computer (SMC). For more information see:

  – Chapter 11: *Payload Development - Software Environment*

Figure 1.1. Wave Glider System (Float, Umbilical, and Sub)

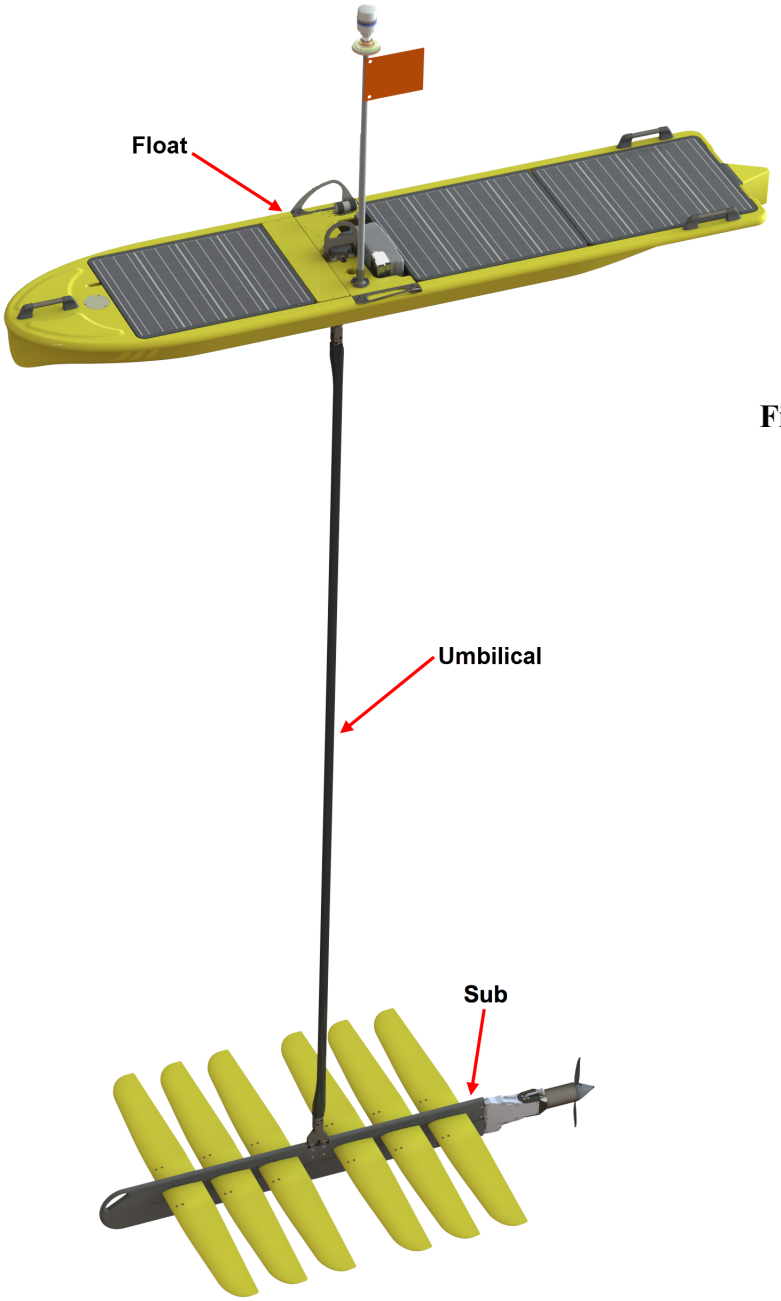


Figure 1.2. Optional Towfish

Liquid Robotics Proprietary and Confidential

Payload Integration

There are a few key parameters that are most likely to determine the feasibility of a payload integration project. Liquid Robotics recommends evaluating a payload integration project with respect to the following key parameters first:

- **Total weight of the payload**
- **Sizes of payload components**
- **Payload mounting requirements**
- **Payload power consumption**

For more information see Chapter 9: *Payload Development Mechanical and Electrical Considerations*. If a payload integration project can be adapted within these key parameters, its feasibility is very likely. The other Wave Glider payload integration parameters are typically easier to comply with.

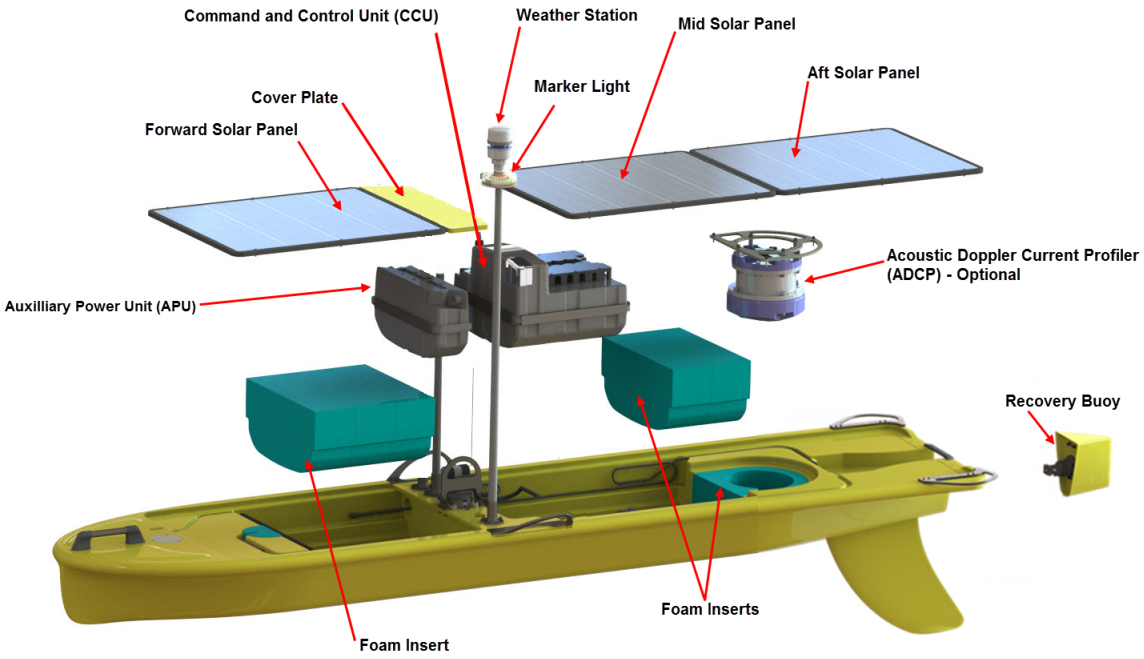
Chapter 2. Mechanical Specifications

This chapter provides mechanical specifications on the Float, CCU, Umbilical, Sub and Towfish options. The drawings in this chapter show dimensions in millimeters and [inches].

Float Assembly

The Float assembly is illustrated in Figure 2.1.

Figure 2.1. Float Assembly Exploded View



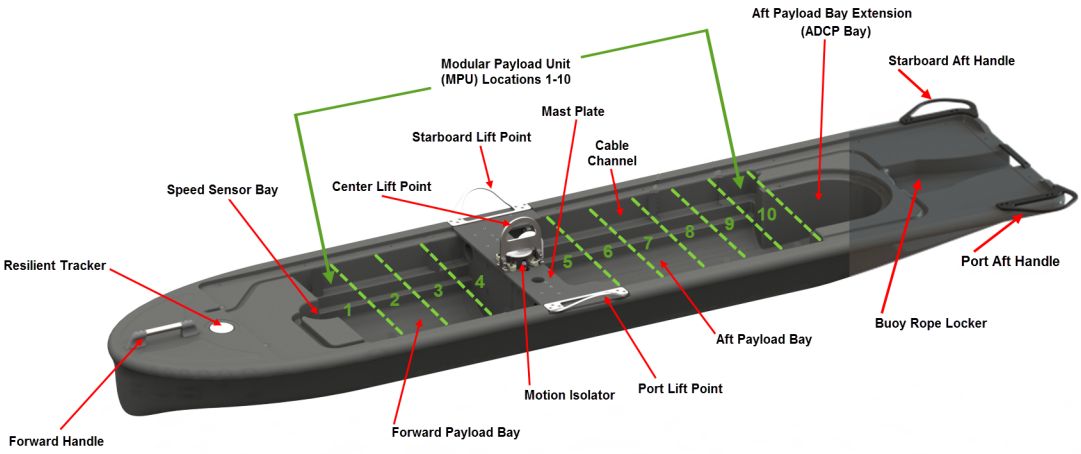
- **Command and Control Unit (CCU)** - Contains the Vehicle Management Computer, standard electronics, and standard batteries. See "Command and Control Unit (CCU)" (page 8) for more information.
- **Auxiliary Power Unit (APU)** - Contains a supplemental battery pack with the same specifications as the battery pack included in the CCU. See "Auxiliary Power Unit (APU)" (page 185) for more information.
- **Forward, Mid, and Aft Solar Panels** - Three identical photovoltaic panels used to charge the on-board batteries.
- **Optional Acoustic Doppler Current Profiler (ADCP)** - An ADCP is used in a wide range of Wave Glider configurations. The Wave Glider Float includes an Aft Payload Bay Extension designed to accommodate an ADCP.
- **Weather Station** - Provides dynamic weather data for the Wave Glider location.
- **Marker Light** - Can be turned on and off manually from WGMS, or other Remote Applications, to improve Wave Glider visibility in low-light conditions.

Liquid Robotics Proprietary and Confidential

- **Cover Plate** - Covers the Modular Payload Unit (MPU) between the Forward Solar Panel and the Mast Plate. In the illustration above, an Auxiliary Power Unit (APU) is located in the MPU directly beneath the Cover Plate. See Figure 2.2 for information on MPUs.
- **Recovery Buoy** - Used to recover the Wave Glider. See "Recovery Buoy and Release Mechanism" (page 6) for more information.
- **Skeg** - Stabilizes the Float in yaw.
- **Foam Inserts** - Cut into 1 MPU size sections to fill unused payload bay space. There is also a foam insert designed to fill space around an ADCP when installed in the Aft Payload Bay Extension.

The Float has some integral features that should be noted:

Figure 2.2. Float Integral Features



- **Forward Handle** - For lifting or guiding the Float in water or when it is on a wheeled stand. Not for hoisting the Wave Glider system (Float + Umbilical + Sub) out of the water. This handle is not load rated.
- **Resilient Tracker** - Provides a secondary source of GPS location information transmitted over the Iridium network, in addition to the tracker included in the CCU Dry Box.
- **Speed Sensor Bay** - Designed to accommodate a speed sensor that is included with the Wave Glider SV3.
- **Modular Payload Unit (MPU) Locations** - An MPU is equal to the size of the smallest modular payload dry box that Liquid Robotics offers, known as a 1-MPU. The Wave Glider Float payload bays accommodate a total of 10 MPUs, 4 in the Forward Payload Bay and 6 in the Aft Payload Bay. The CCU is a 3-MPU that typically occupies MPU locations 5, 6 and 7 directly aft of the Mast Plate. See "Modular Payload Dry Boxes" (page 180) for more information.
- **Forward Payload Bay**- Located between the Speed Sensor Bay and the Mast Plate. Able to accommodate 4 MPUs.
- **Port, Starboard, and Center Lift Points** - For lifting (hoisting) the Wave Glider system (Float, Umbilical, and Sub) out of the water.
- **Motion Isolator** - Provides a shock absorbing Umbilical-to-Sub attachment.

Liquid Robotics Proprietary and Confidential

- **Mast Plate** - Includes several threaded titanium mounting points for masts or sensors.
- **Aft Payload Bay** - Located between the Mast Plate and the Aft Payload Bay Extension. Able to accommodate 6 MPUs.
- **Aft Payload Bay Extension** - Designed to accommodate an optional Acoustic Doppler Current Profiler (ADCP) - can be used for other payloads, if an ADCP is not installed.
- **Cable Channel** - A cavity for running cables along the sides of the Forward and Aft Payload Bays.
- **Buoy Rope Locker** - Provides storage for the rope attached to the Recovery Buoy. See "Recovery Buoy and Release Mechanism" (page 6) for more information.
- **Port and Starboard Aft Handles** - For lifting or guiding the Float in water or when it is on a wheeled stand. Not for hoisting the Wave Glider system (Float + Umbilical + Sub) out of the water. These handles are not load rated.

Recovery Buoy and Release Mechanism

To facilitate recovery of the Wave Glider system, a buoy with 12.192 m [40 ft] of floating, non-fouling line attached to it can be deployed from the aft end of the Float. The line is stored in a Line Locker below the aft solar panel.

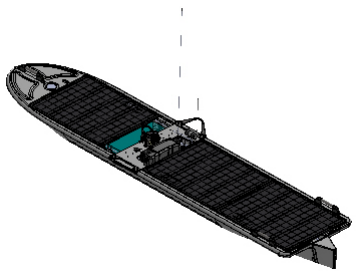
Two spring loaded latches retain the buoy by clasping a latch plate that extends from the buoy. When the release command is issued, a linear actuator forces a piston to open the latches and continue extending to push on the latch plate, until the buoy is released from the Float. The pushing force from the linear actuator will overcome any potential bio fouling.

When the buoy is released, drag on the buoy from water resistance will pull the line out of the line locker. A grappling hook can be used to capture the red and yellow colored line trailing the float, so you can bring the Wave Glider vehicle into position for hoisting.

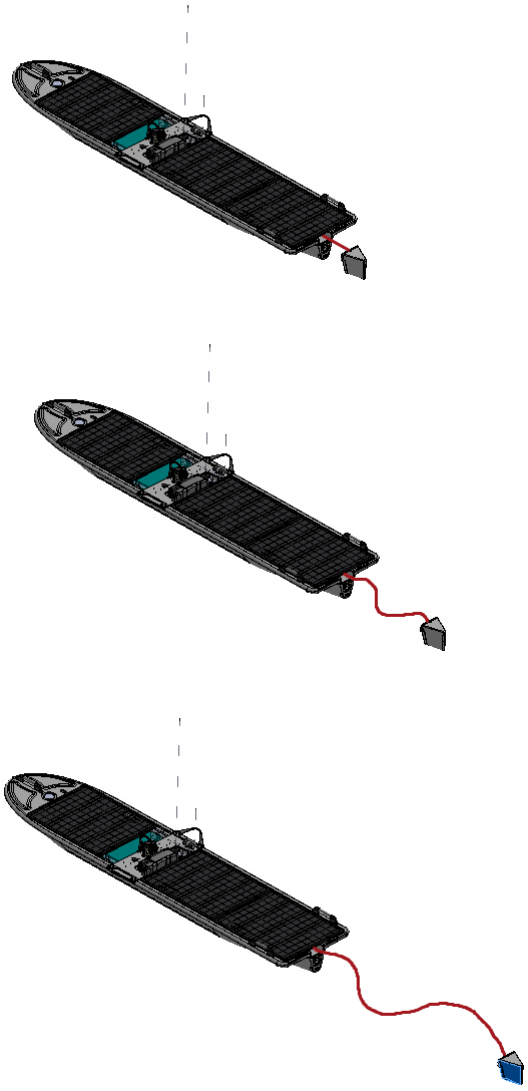
Important

Do not use the Recovery Buoy or the the red and yellow line attached to the Recovery Buoy to hoist the Wave Glider vehicle, it will not safely support the weight. Refer to the *Wave Glider® SV3 Launch and Recovery Guide* for more information.

Buoy Release Sequence:



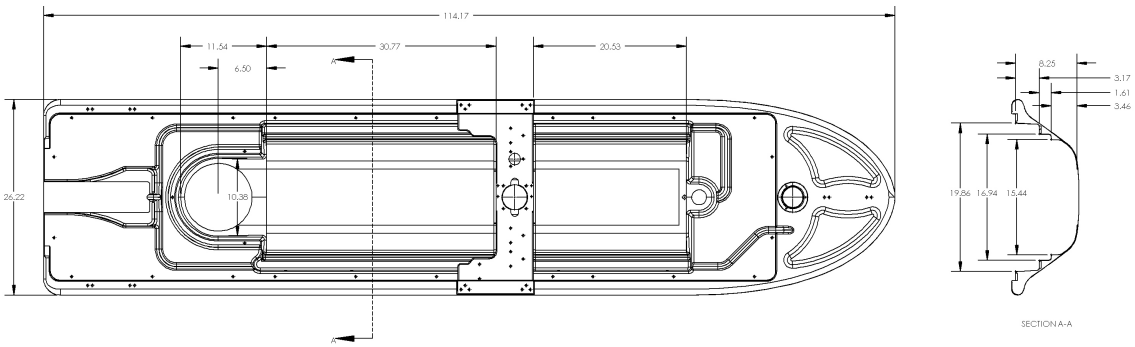
Liquid Robotics Proprietary and Confidential



The Recovery Buoy can be manually reloaded into the Wave Glider Float in the field without tools. Also, if necessary, the Recovery Buoy can be manually released from the Float by inserting a 6" screw driver or similar object into a hole in the Recovery Buoy and pressing to release the latch.

Float Overall Dimensions

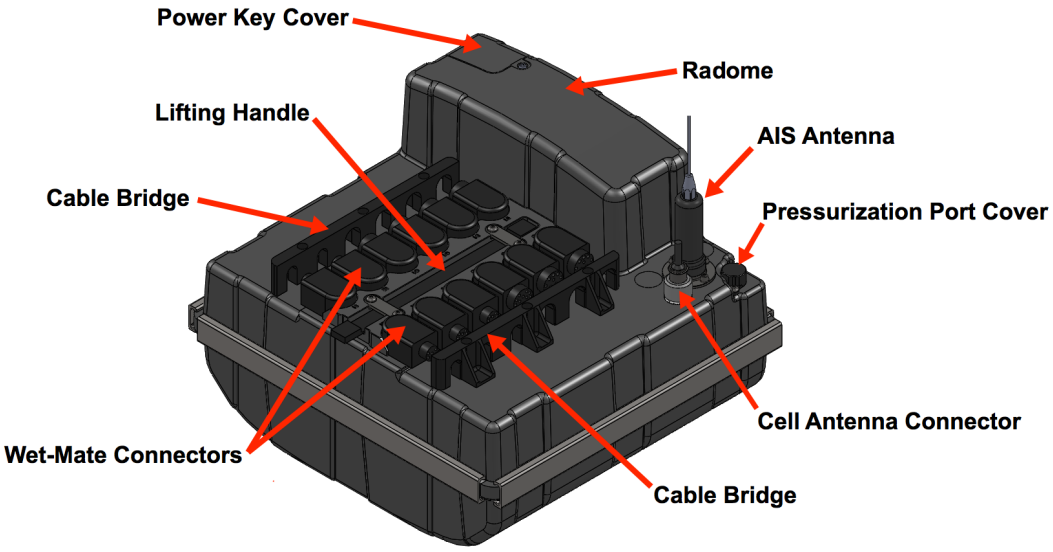
Figure 2.3. Float Overall Dimensions (mm)



Command and Control Unit (CCU)

The CCU is a factory sealed module that is mounted in the Float at the forward end of the aft payload bay, with its raised Radome tight to the mast plate. It contains the computing, control and communications electronics, and the batteries that store energy from the solar panels. Additional electronics can be added to optional Liquid Robotics Modular Payload Dry Boxes that typically interface with the CCU. For more information see "Modular Payload Dry Boxes" (page 180). **The CCU contains no customer-accessible components.**

Figure 2.4. CCU



The main external components of the CCU are:

- **Power Key Cover** - Removable cover that protects the Power Key which is the main power on/off switch for the Wave Glider electrical system. See "Power Key" (page 34)for more information.
- **Radome** - GPS, Iridium, 2.4 GHz WiFi and 5 GHz WiFi antennas are contained in the Radome.

- **AIS Antenna** - Receives Automatic Identification System reports from class A and class B vessels in the vicinity of the Wave Glider. These reports provide vessel ID, position, heading and other standard AIS data.
- **Pressurization Port Cover** - Protective cover for the dry box pressurization valve.
- **Cell Antenna Connector** - For connecting a cell antenna to be used with the cell modem in the CCU.
- **Wet-Mate Electrical Connections** - Twelve watertight electrical connectors for on-board devices, payloads, and debugging. For more information on the CCU connectors and pin outs, see "CCU Electrical" (page 32) and "CCU Standard Electrical Connections" (page 41).
- **Cable Bridge** - Keeps wet-mate connectors firmly in place.
- **Lifting Handle** - Nylon strap for lifting the CCU out of the payload bay.

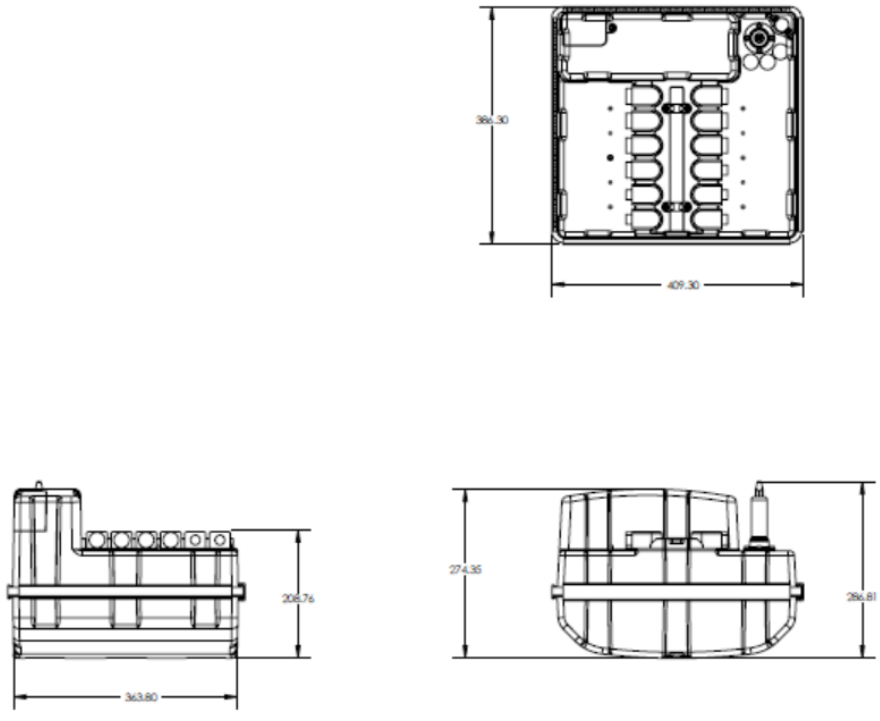
For information on CCU power consumption, see " Batteries" (page 66).

For information on CCU Communications Channels, see Chapter 4: *CCU Remote Communications and Control*.

CCU Weight and Dimensions

The CCU assembly weighs 17.0 Kg (37.5 lb). The dimensions of the CCU are shown in Figure 2.5.

Figure 2.5. CCU Dimensions (mm)



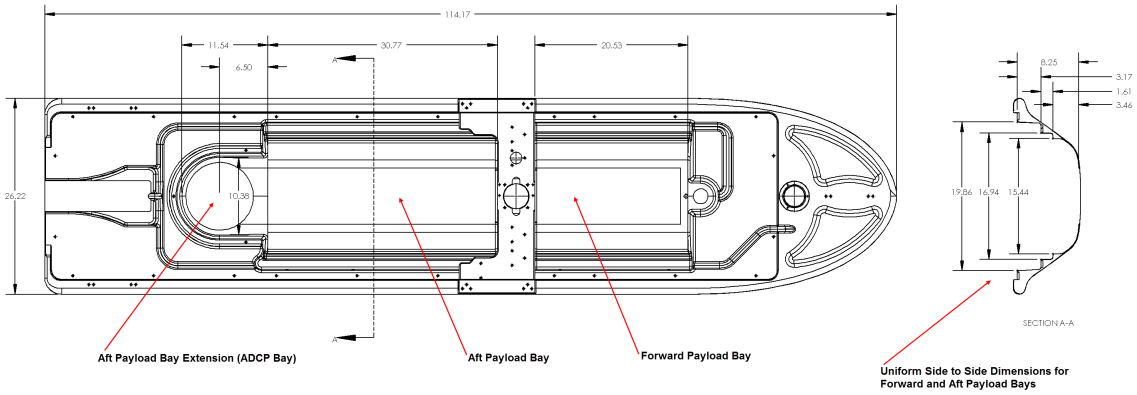
Float Payload Bays

The Wave Glider SV3 Float payload bays vary in length in the forward to aft dimension, but they have uniform dimensions from side to side. This design supports the use of the Liquid Robotics Modular

Payload Dry Boxes that are available in lengths of one Modular Payload Unit and three Modular Payload Units. A Modular Payload Unit (MPU) is equal to the space that an Liquid Robotics 1-MPU Payload Dry Box takes up in the payload bay (133 mm in the forward to aft dimension). The SV3 Forward Payload Bay has a capacity of 4 MPUs. The SV3 Aft Payload Bay has a capacity of 6 MPUs. For example, an Liquid Robotics 3-MPU payload dry box takes up 3-MPUs of space in an SV3 payload bay. The Float CCU takes up 3-MPUs, and an Auxiliary Power Unit (APU) takes up 1-MPU. See Figure 2.2 and "Modular Payload Dry Boxes" (page 180) for more information.

The Aft Payload Bay Extension (ADCP Bay) extends from the aft end of the Aft Payload Bay. This bay was designed to house a Teledyne RD Instruments acoustic doppler current profiling sensor (ADCP), however, it is available for other uses if an ADCP is not installed.

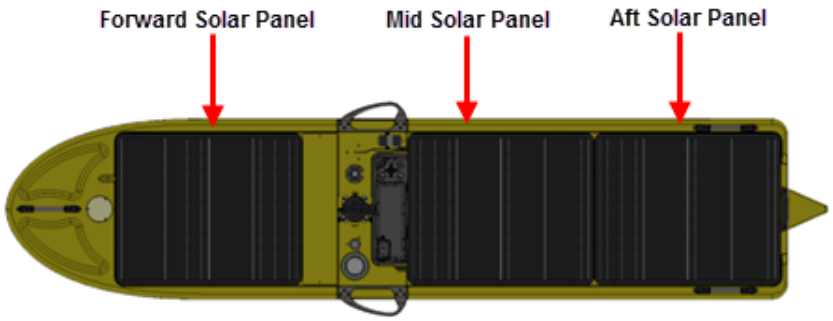
Figure 2.6. Float Payload Bay Dimensions [inches]



Solar Panels

This section describes the features of the standard Wave Glider solar panels.

Figure 2.7. Standard Solar Panels



Liquid Robotics Proprietary and Confidential

Solar Panel Specifications

Table 2.1. SV3 Solar Panel Specifications

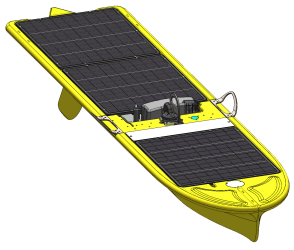
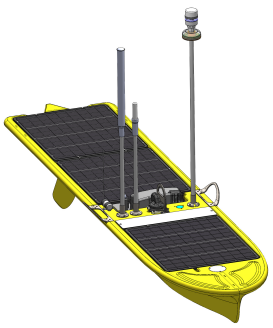
Panel Power	52 W
Panels per Vehicle	3
Vehicle Power	156 W
Construction	Fiberglass and foam laminate with hard points
Top Sheet	Glass

Shading

Shading in general comes from environmental conditions such as cloud cover, and from system components mounted on the vehicle. Shading from environmental conditions is typically temporary and is beyond our control. To optimize total power input, all sensors would have to be mounted below the solar panels to prevent shading. Since this is typically not practical, the following recommendations should be followed to minimize shading.

- If a tall object such as a mast is required, attempt to design it to be narrower than a single cell. This will not eliminate shading, but it reduces the shading impact.
- If an object is wide, attempt to design it to be as short as possible. With this design, shading is not a problem during the middle of the day.
- To maximize power harvest when there are typical shading elements on a vehicle (mid-mounted antenna masts etc.), a course that is predominantly perpendicular to the sun's rays will be best. During early morning and late evening, a North-South course line will minimize solar shadowing. In late morning and late afternoon, a NE-SW and SE-NW course line can be advantageous. During mid-afternoon, an East-West course line will result in the best solar harvest. Early morning and late afternoon solar harvest will be relatively less than mid afternoon solar harvest, regardless of the heading. For this reason, following a course to maximize solar harvest will be most effective during the hours immediately before, and immediately after, noon (Wave Glider local time). During periods of heavy cloud cover, solar shading is relatively insignificant; solar harvest is primarily limited by low ambient light intensity.

Table 2.2. Solar Panel Shading Considerations

Minimal Shading Potential	Typical Shading Potential
	

Liquid Robotics Proprietary and Confidential

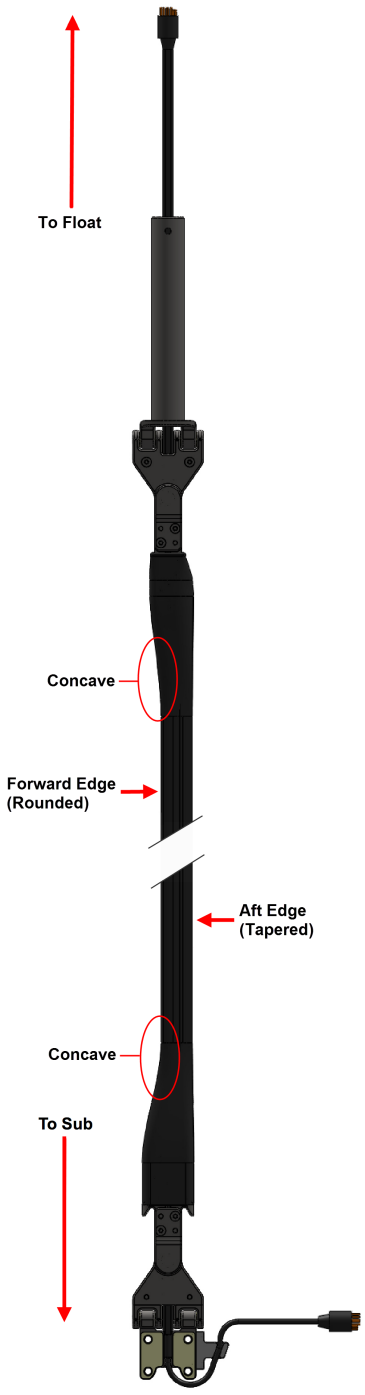
Float to Sub Umbilical

The Umbilical (also referred to as the "Tether") is a flexible cable that connects the Float to the Sub. The standard SV3 Umbilical is 8 m [26.2467 ft] long and streamlined to reduce drag.

The Umbilical contains high-strength reinforcement cables designed to support the weight of the Sub and the tension the Sub exerts as it responds to wave motion. The Umbilical also contains electrical wiring for power, Thrudder control, auxiliary propulsion, and payload communications. The pin-out for the Umbilical connector to the CCU and the Umbilical connector to the Thrudder Module provide details on the wiring contained in the Umbilical. For more information see “U1 - Umbilical Connector” (page 45) and “Thrudder Module Umbilical Connector” (page 63).

Liquid Robotics Proprietary and Confidential

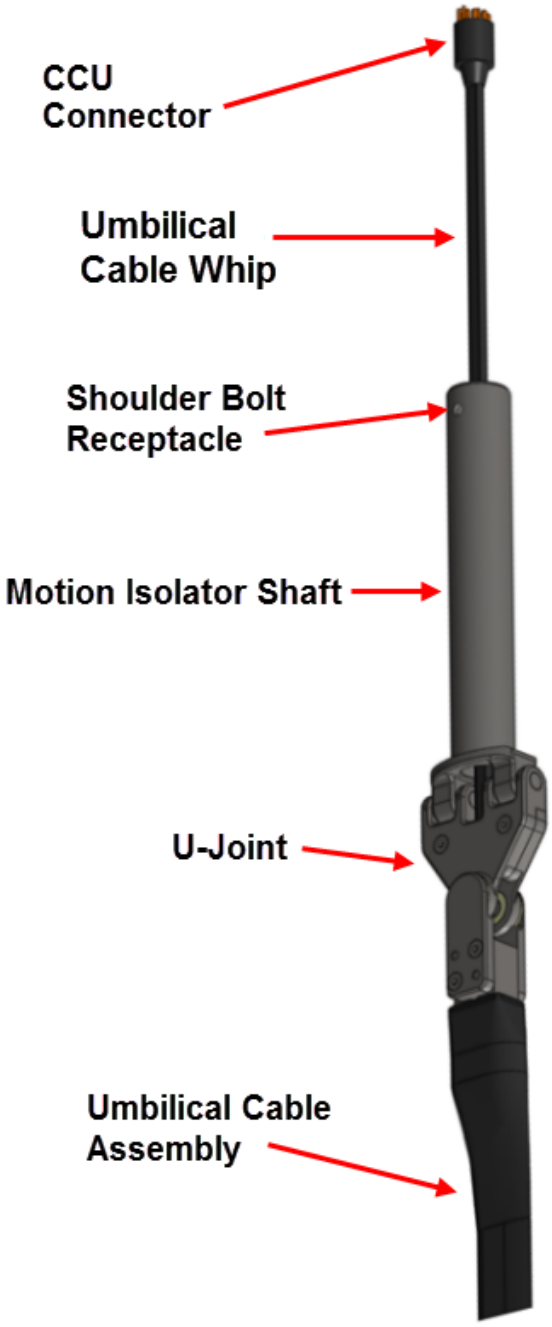
Figure 2.8. Umbilical Assembly Overview



- **Forward Edge** - For streamlining, the forward edge is rounded. Note that the forward edge is also slightly concave at each end.
- **Aft Edge** - For streamlining, the aft edge is tapered.

Liquid Robotics Proprietary and Confidential

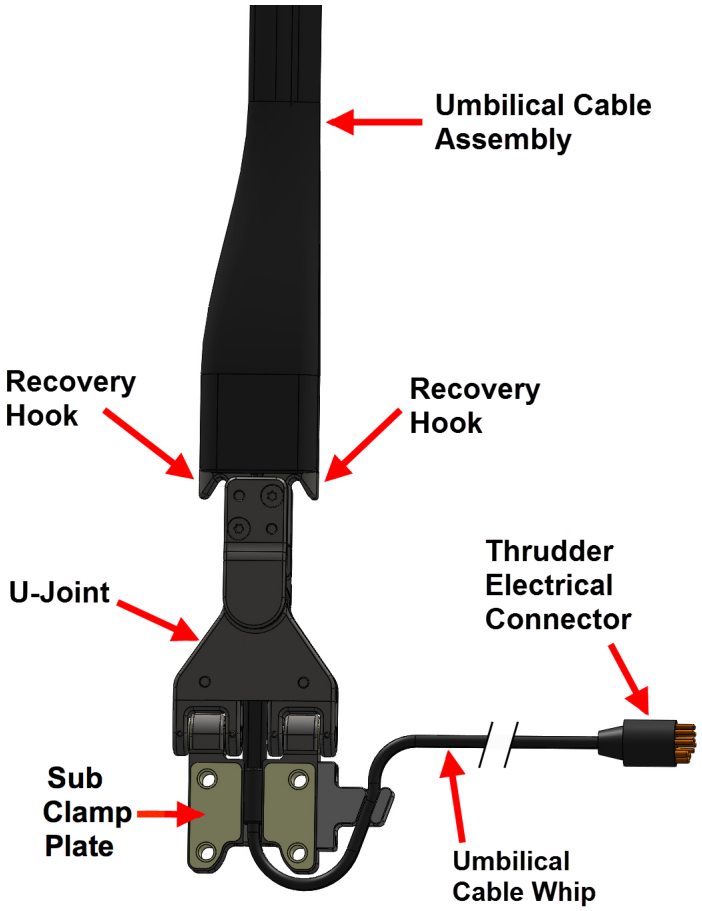
Figure 2.9. Umbilical-to-Float Connection



- **Float CCU Connector** - Watertight power and communications connection between the CCU and the Umbilical.
- **Umbilical Cable Whip** - An extension of the Umbilical power and communications cable that extends from the Motion Isolator assembly to the CCU when the Umbilical is attached to the Float.

- **Shoulder Bolt Receptacle** - Where the Shoulder Bolt passes through the Motion Isolator Shaft locking it into the Motion Isolator assembly.
- **Motion Isolator Shaft** - Provides an attachment between the Float and the Umbilical when inserted into the Motion Isolator and secured with the Shoulder Bolt.
- **U-Joint** - Provides two axis of rotation to prevent stress on the Umbilical.
- **Umbilical Cable Assembly** - Attaches the Float to the Sub while providing power and communications connectivity.

Figure 2.10. Umbilical-to-Sub Connection



- **Umbilical Cable Assembly** - Attaches the Float to the Sub while providing power and communications connectivity.
- **Recovery Hooks** - Provide a positive locking feature for the Sub Recovery Mechanism.
- **U-Joint** - Provides two axis of rotation to prevent stress on the Umbilical.
- **Umbilical Cable Whip** - An extension of the Umbilical power and communications cable that extends to the Thrudder Module when the Umbilical is attached to the Sub.
- **Thrudder Electrical Connector** - Watertight power and communications connection between the Umbilical and the Thrudder Module.

Liquid Robotics Proprietary and Confidential

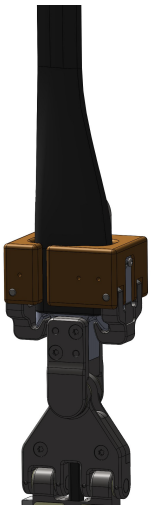
- **Sub Clamp Plate** - The mechanical connection to the Sub - attaches to the Sub frame with 4 bolts.

Sub Recovery Mechanism

The Sub Recovery Mechanism is used to hoist the Sub out of the water. A line is attached to the ring on the Sub Recovery Mechanism, then the Sub Recovery Mechanism is slid over the Umbilical and dropped so it locks onto the Recovery Hooks at the bottom of the Umbilical. For detailed Sub Recovery instructions see the Liquid Robotics Launch and Recovery Procedures for the Wave Glider SV3.

Figure 2.11. Sub Recovery Mechanism

Figure 2.12. Sub Recovery Mechanism Locked Onto Recovery Hooks

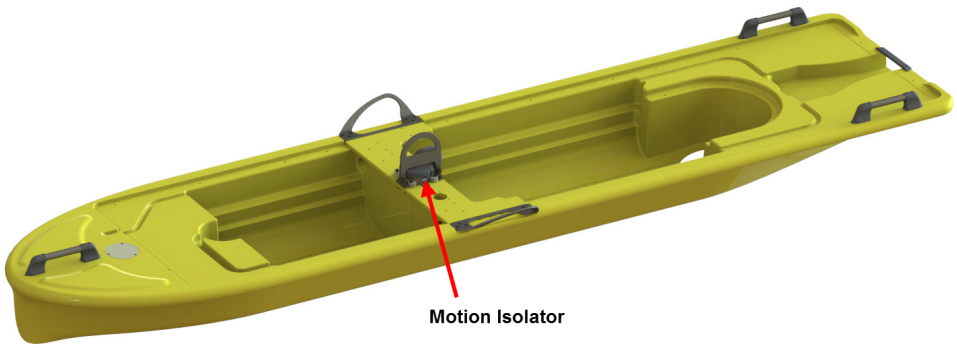


Liquid Robotics Proprietary and Confidential

Motion Isolator

The Motion Isolator provides a shock absorbing Umbilical-to-Sub attachment. The Motion Isolator assembly is located directly under the Center Lift Point on the Mast Plate of the Float.

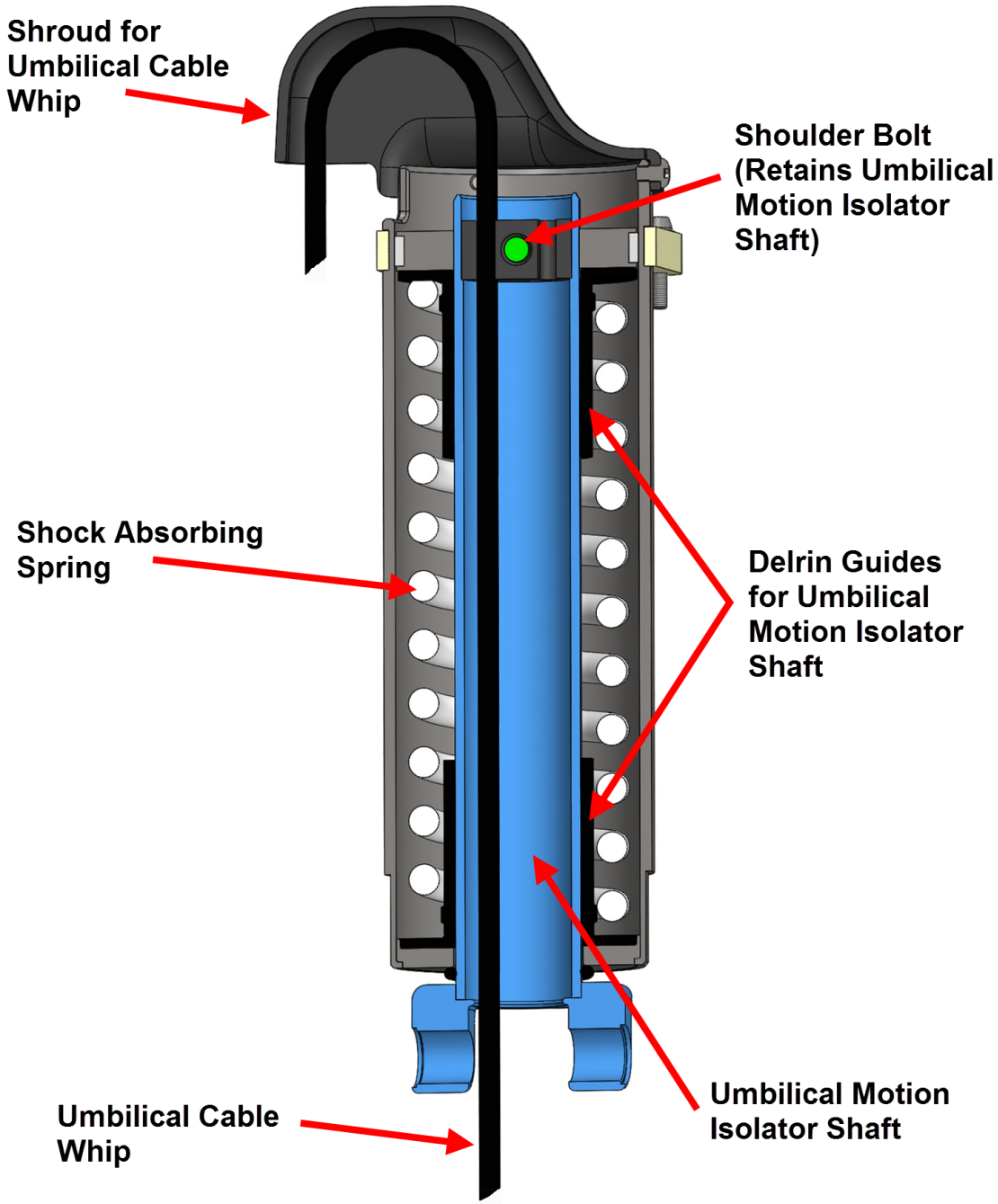
Figure 2.13. Motion Isolator - Top View



The Motion Isolator Shaft on the Umbilical assembly inserts into the Motion Isolator assembly and is locked in place with a Shoulder Bolt that passes through the Motion Isolator Shaft. The spring in the Motion Isolator assembly absorbs shock in rough water and high seas. The Umbilical Cable Whip passes through the protective Shroud and continues on to the CCU.

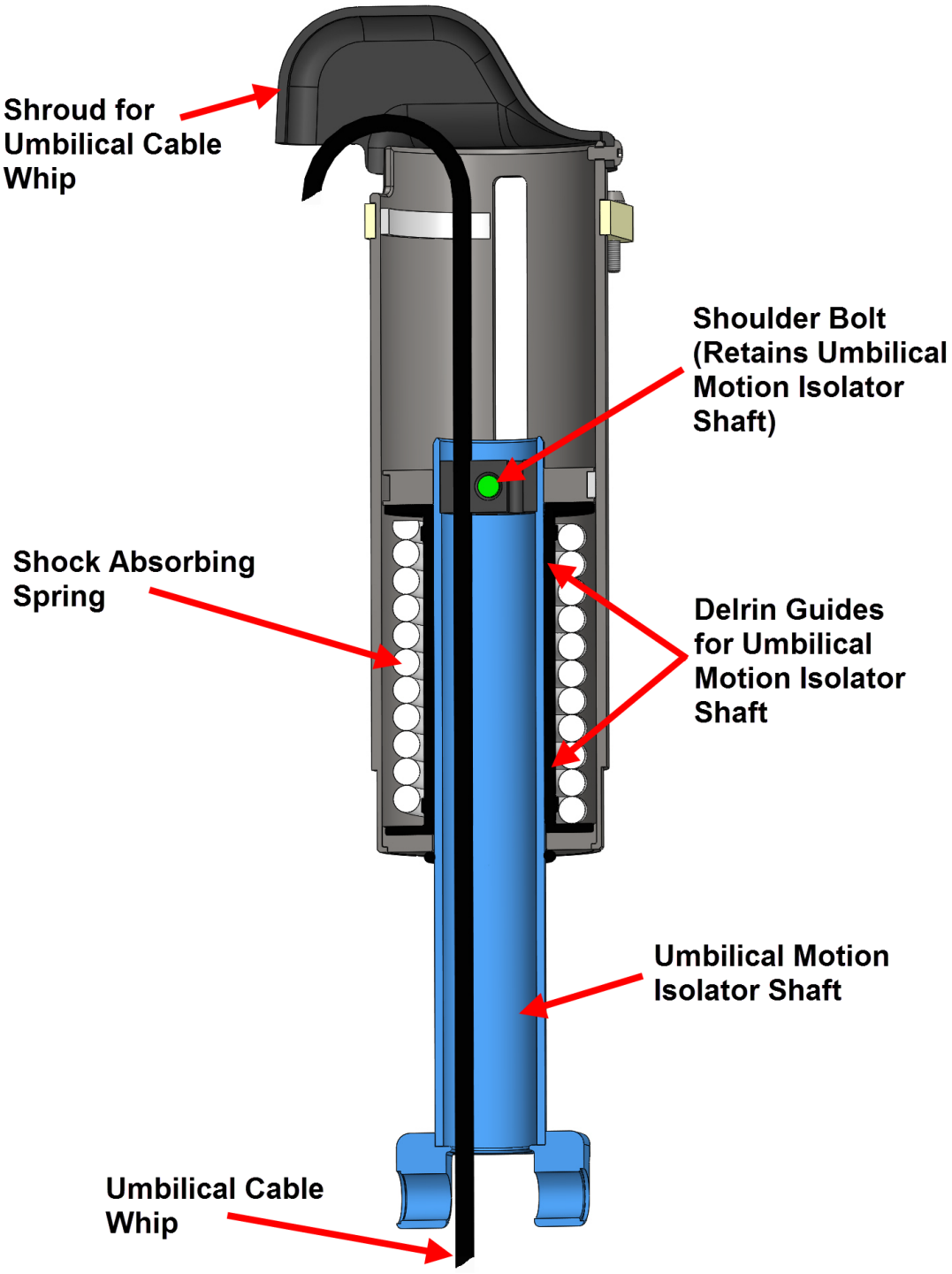
Liquid Robotics Proprietary and Confidential

Figure 2.14. Motion Isolator Extended, Cut-Away View



Liquid Robotics Proprietary and Confidential

Figure 2.15. Motion Isolator Compressed, Cut-Away View

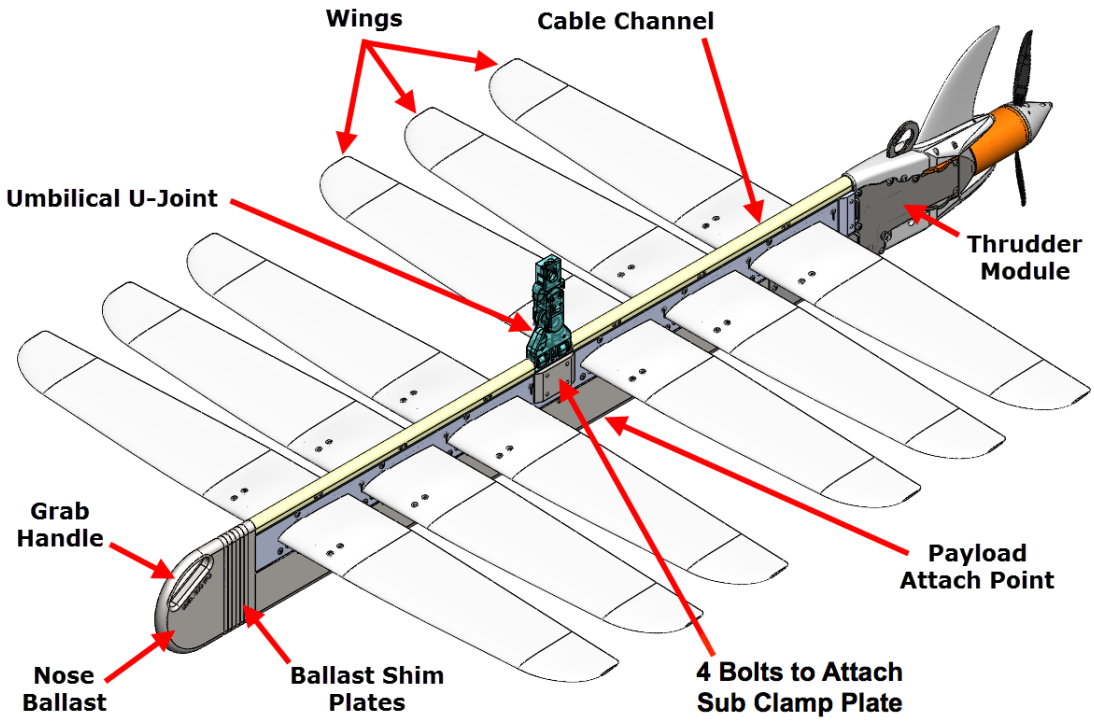


Liquid Robotics Proprietary and Confidential

Sub

The Sub converts wave motion into forward thrust. The attached Thrudder Module steers the Wave Glider system and also provides supplemental thrust on demand with a motorized propeller. The Sub can be used to carry some underwater payloads, although use of a Towfish for these payloads is recommended. See "Towfish Options" (page 26) for more information.

Figure 2.16. Sub



- **Grab Handle** - Used for launch and retrieval.
- **Nose Ballast** - Forward-located ballast that also serves as a Grab Handle for the Sub.
- **Ballast Shim Plates** - Can be added or removed to compensate for payload mounting that impacts the Sub center-of-gravity. Consult with Liquid Robotics before attempting to mount a payload on the Sub or adjusting the Ballast Shim Plates.
- **Umbilical U-Joint** - Provides two axis of rotation to prevent stress on the Umbilical.
- **Wings** - Six pairs of wings provide thrust. Each left-right pair of Wings rotates independently from the other five wing pairs, but the two wings in each left-right pair are attached and rotate in unison.
- **Cable Channel** - A fiberglass shroud provides a narrow protected channel along the top of the Sub frame. The Umbilical Cable Whip runs through the Cable Channel to get to the water-tight connector on the Thrudder Module.
- **Thrudder Module** - Factory sealed. Contains the servo motor and mechanism for pivoting the Fin and the auxiliary Thruster to steer the Wave Glider. It also contains a compass. The Thrudder Module has 3 water-tight electrical connectors for the Umbilical Cable Whip, a Sub or Towfish payload, and a connector for a remote compass.

Liquid Robotics Proprietary and Confidential

- **Sub Clamp Plate Attachment Bolts** - 4 bolts that attach the Umbilical assembly to the Sub.
- **Payload Attach Point** - 2 bolt holes for attaching a payload. There is an adjacent electrical channel for snaking a power and communications line between the payload and the payload connector on the Thrudder Module. See "Sub Dimensions and Payload Attachment Points" (page 23) for more information.

Figure 2.17. Sub with Thrudder Turned - Side View

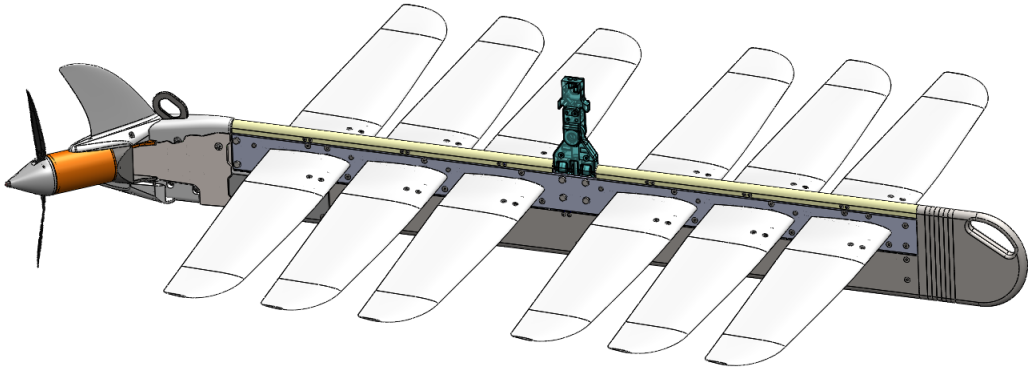
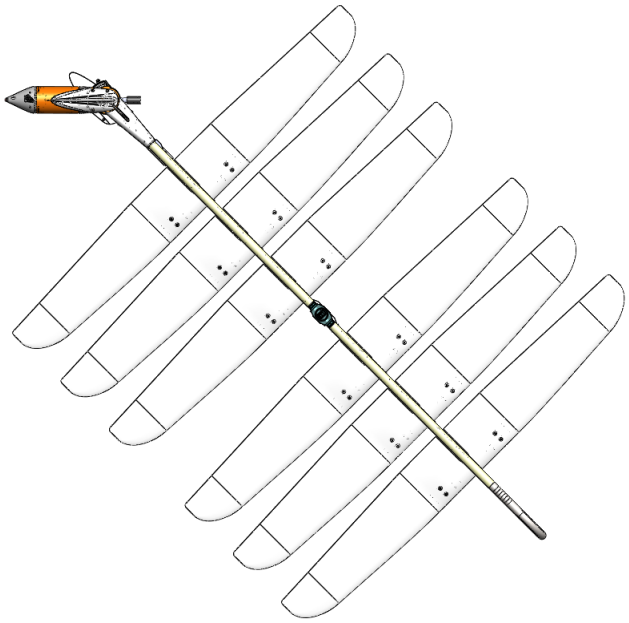


Figure 2.18. Sub with Thrudder Turned - Top View



When the Sub is carrying payloads, streamlined fairings can be fabricated to cover the payload. An example is shown in the figures below. The actual shape of the cover can vary depending on the type, shape and volume of the payload.

Liquid Robotics Proprietary and Confidential

Figure 2.19. Sub Payload With Custom Fairing

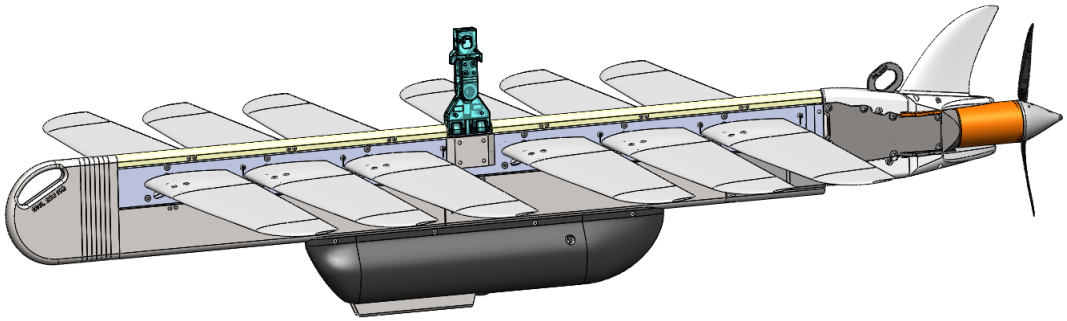
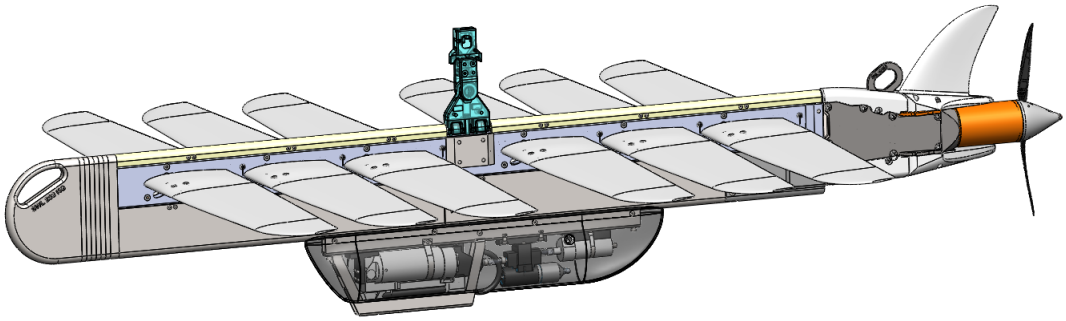


Figure 2.20. Sub Payload With Custom Fairing - Translucent



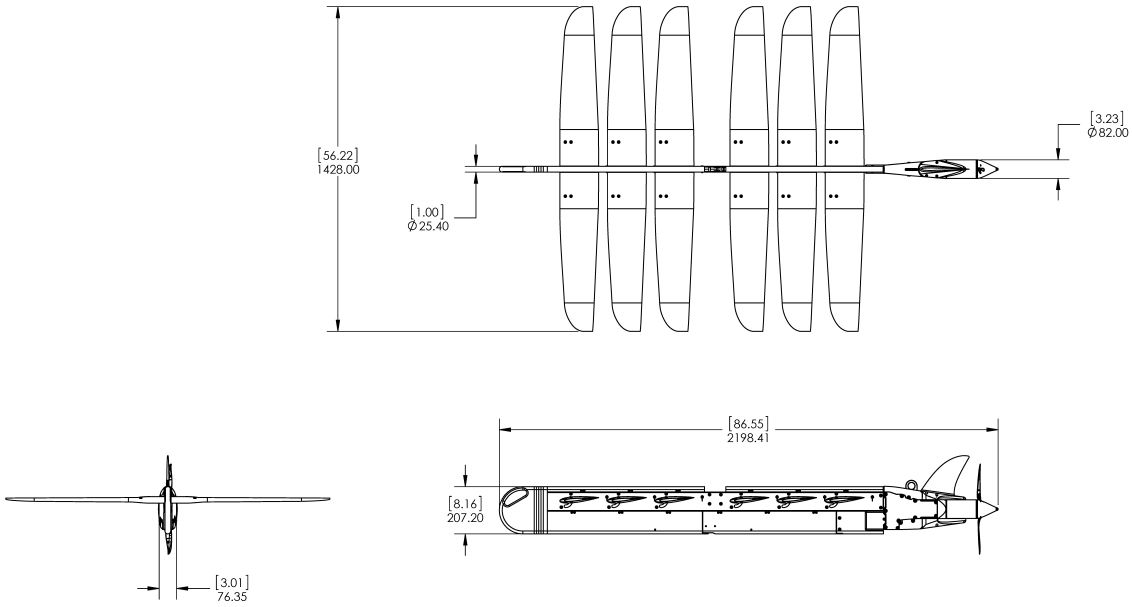
See "Sub Payload Considerations" (page 191) for additional design considerations for Sub payloads.

Liquid Robotics Proprietary and Confidential

Sub Dimensions and Payload Attachment Points

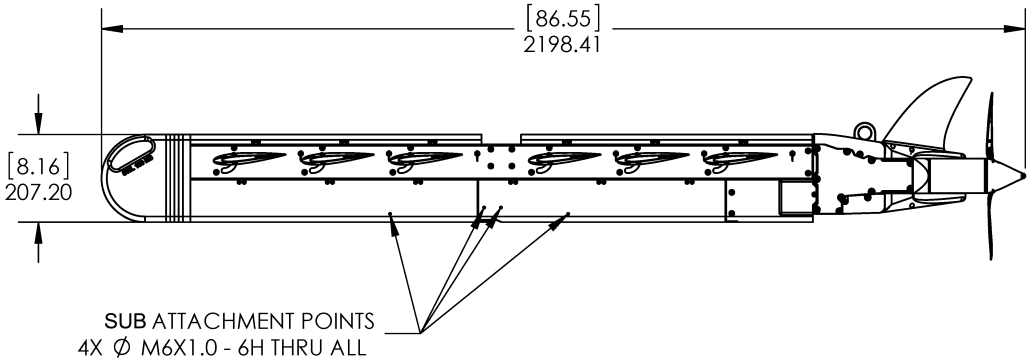
The Sub assembly overall dimensions and attachment points are detailed in the following drawings.

Figure 2.21. Sub Overall Dimensions - mm [inch]



The attachment points consist of 4 threaded holes through the Sub Frame. There is also an opening in the Sub Frame where cables can be routed through the Sub Frame to the Expansion Port on the Thrudder assembly. Typically these attachment points are used for a towing bracket where a Towfish tether is attached or for a Sub-mounted payload and the fairings that cover it.

Figure 2.22. Sub Attachment Points mm [inch]

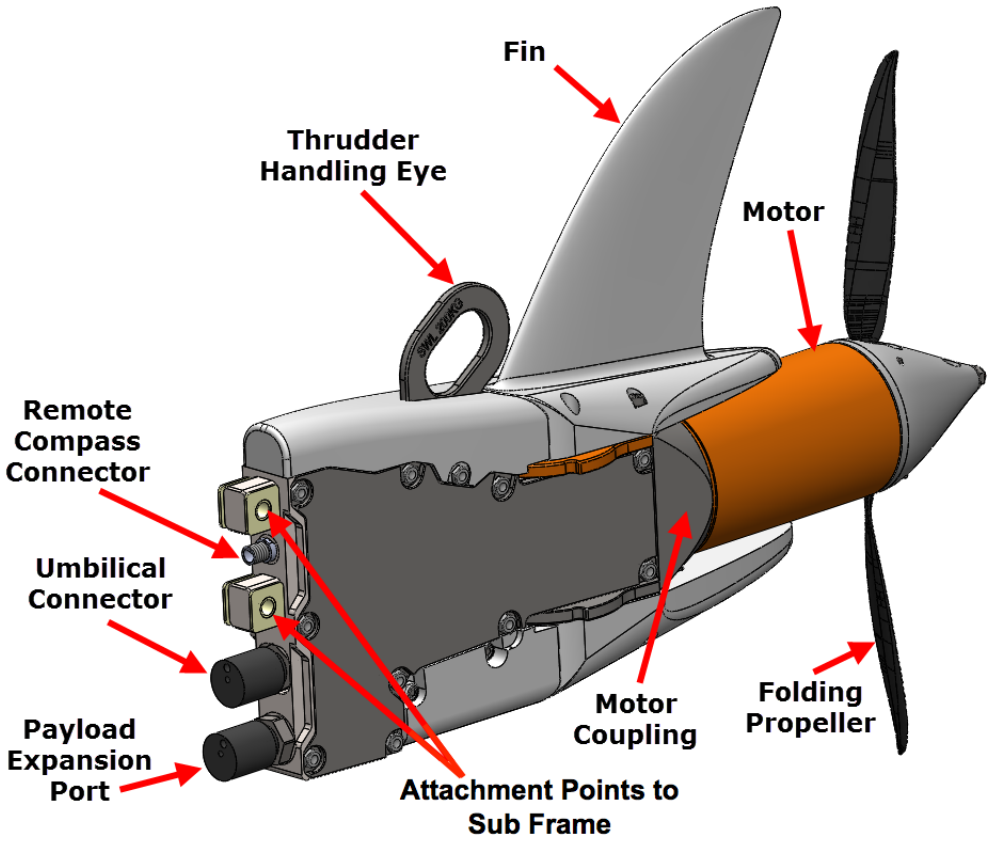


Liquid Robotics Proprietary and Confidential

Thrudder Module

The Thrudder Module is the core component in compass based navigation, a control point for Sub and Towfish payloads, and a source of auxiliary propulsion when needed. The Thrudder Module is attached to the Sub frame with two bolts. The Fin and motorized propeller are attached to the pivoting Motor Coupling to steer the Sub.

Figure 2.23. Thrudder Module



Note

The Thrudder Handling Eye is a marine operations safety feature for stabilizing the Sub assembly with a boat hook to keep it from rotating when it is hoisted out of the water. It can also be used to hoist the Thrudder Module, but it cannot be used for hoisting the entire Sub assembly.

There are also two fairings that enclose and protect cables and electrical connections. See “Thrudder Module Electrical Connections” (page 61) for electrical specifications.

Liquid Robotics Proprietary and Confidential

Figure 2.24. Thrudder Attachment Exploded View

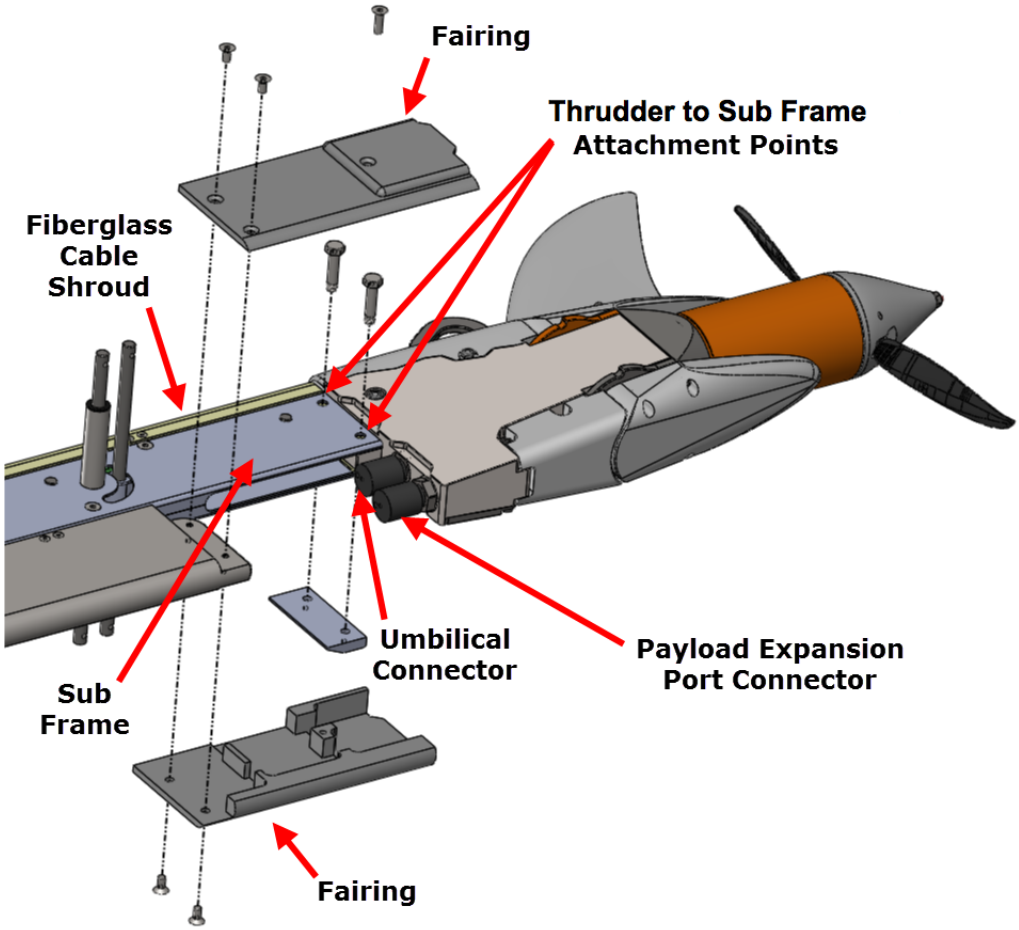
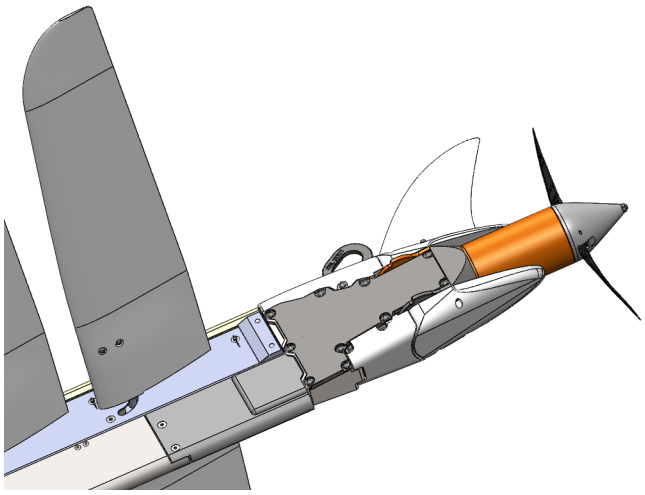


Figure 2.25. Thrudder Attached to Sub with Fairings Installed



Liquid Robotics Proprietary and Confidential

Towfish Options

Towfish are sub-sea extentions of the Wave Glider system. A Towfish is tethered to a specific attachment point on the Sub that prevents the tow cable from interfering with the Thruster propeller. The tow cable includes wiring that provides a communication gateway for data transmissions between towfish-mounted sensors, and data gathering on the Float, or on shore. Liquid Robotics Towfish support both flooded and dry payloads and can be equipped with electronics to manage multiple sensors on the same Towfish. This section provides an overview of the Wave Glider Towfish options. For more information on Towfish sensor integration parameters, contact Liquid Robotics.

There are two standard Towfish options for the Wave Glider:

- **Gateway Towfish** For information on the Gateway Towfish see “Gateway Towfish” (page 26).
- **PAM Buoy Towfish** For information on the PAM Buoy Towfish see “PAM Buoy Towfish” (page 27).

Gateway Towfish

The Gateway Towfish provides a dynamically isolated location to mount an acoustic transducer and accompanying electronics for communication with sub-sea sensors. It is a streamlined, buoyancy neutral vessel that is tethered to the Sub with a shock-absorbing tow cable that has integrated power and communications wiring. The tow cable connects the Gateway Towfish electrically to the Wave Glider CCU. The CCU provides power and facilitates communications. The Towfish sends and receives data over the tow cable, through the Thrudder Module, to the CCU, where it can be stored or transmitted to shore over Iridium Satellite or cellular communications channels. If there is a Float mounted payload associated with the acoustic transducer on the Towfish, data is transmitted from the Towfish to the CCU, to the Float-mounted payload, then back to the CCU for storage or transmission to shore.

Figure 2.26. Gateway Towfish

The Gateway Towfish is designed to carry an acoustic transducer and a modem, with related electronics located in a payload dry box on the Float. Its payload bay can accept a 4.5" diameter transducer weighing up to 7 lbs. and it is designed to accept the Benthos modem or the WHOI micro-modem, with up to 3.5 lbs of total modem weight. Once the contents of the payload are installed, the Towfish must be adjusted to neutral buoyancy and level trim.

Liquid Robotics Proprietary and Confidential

Power requirements depend on the sensor and modem that will be used. A high voltage board (located in a Float payload dry box) may be required to power the electronics in the Towfish. Power available to the payload is dependent on Thrudder Module power limitations, as well as over-all Wave Glider system power limitations.

The Gateway Towfish has been proven in high sea states and has been successful in deep water off shore missions.

PAM Buoy Towfish

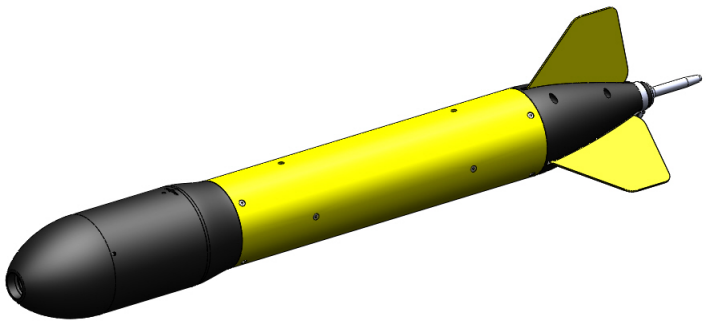
The PAM Buoy Towfish provides a simple cylindrical payload bay with modular nose and tail assemblies. It is a streamlined buoyancy neutral vessel that is tethered to the Sub with a shock-absorbing tow cable with integrated power and communications wiring. The tow cable connects the Gateway Towfish electrically to the Wave Glider CCU. The CCU provides power and facilitates communications. The Towfish sends and receives data over the tow cable, through the Thrudder Module, to the CCU, where it can be stored or transmitted to shore over Iridium Satellite or cellular communications channels. If there is a Float mounted payload associated with the acoustic transducer on the Towfish, data is transmitted from the Towfish to the CCU, to the Float-mounted payload, then back to the CCU for storage or transmission to shore.

The length of the payload bay can be adjusted based on the length of the fiberglass tube that forms the PAM Buoy Towfish body. The PAM Buoy Towfish bay is flooded, but can also carry dry payloads inside an optional sealed enclosure. Inlet and outlet scoops as well as other types of through-wall fittings can easily be integrated into the PAM Buoy Towfish.

The number of payloads the PAM Buoy Towfish can support depends on the weight, physical dimensions, and power requirements of the payloads. Power available to the payloads is dependent on Thrudder Module power limitations, as well as over-all Wave Glider system power limitations.

Once the contents of the payloads are installed, the Towfish must be adjusted to neutral buoyancy and level trim.

Figure 2.27. PAM Buoy Towfish



Liquid Robotics Proprietary and Confidential

Tow Cable

The same 5.75 m (18.86 ft) long tow cable is used to connect the Gateway or PAM Buoy Towfish to the Sub. It is designed with attached floats and weights to absorb shock, protecting the towfish from the surging action of the Sub. The speed of the Wave Glider is reduced by 3 to 5% when a properly configured Towfish is attached.

Figure 2.28. Towfish with Tow Cable 3D

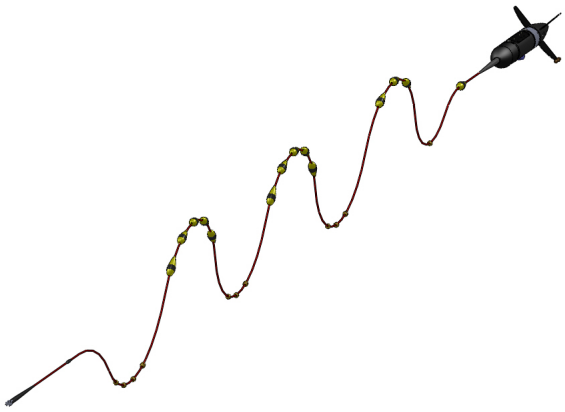


Figure 2.29. Towfish with Tow Cable Side View

Note

If you are planning shallow water operations, be aware that the Towfish flies approximately 1 to 2 meters below the sub.

Liquid Robotics Proprietary and Confidential

Custom Towfish

If a Gateway or PAM Buoy Towfish cannot be adapted to your application, Liquid Robotics may be able to develop a custom Towfish to meet your needs. The following Modular Towfish and Large Payload Towfish are examples of custom Towfish Liquid Robotics has developed in the past.

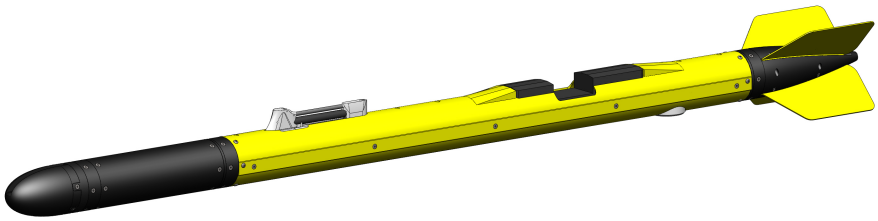
Example: Custom Modular Towfish

The Modular Towfish concept allows for multiple payloads, each with dedicated electrical power and communications provided by the on-board Sensor Management Computer that is connected to the Float CCU through the Thrudder Module and a Float payload dry box.

The number of payloads a Modular Towfish can support depends on the weight, physical dimensions, and power requirements of the payloads. Power available to the payloads is dependent on Thrudder Module power limitations, as well as over-all Wave Glider system power limitations.

Once the contents of the payloads are installed, the Towfish is adjusted to neutral buoyancy and level trim.

Figure 2.30. Example of a Modular Towfish

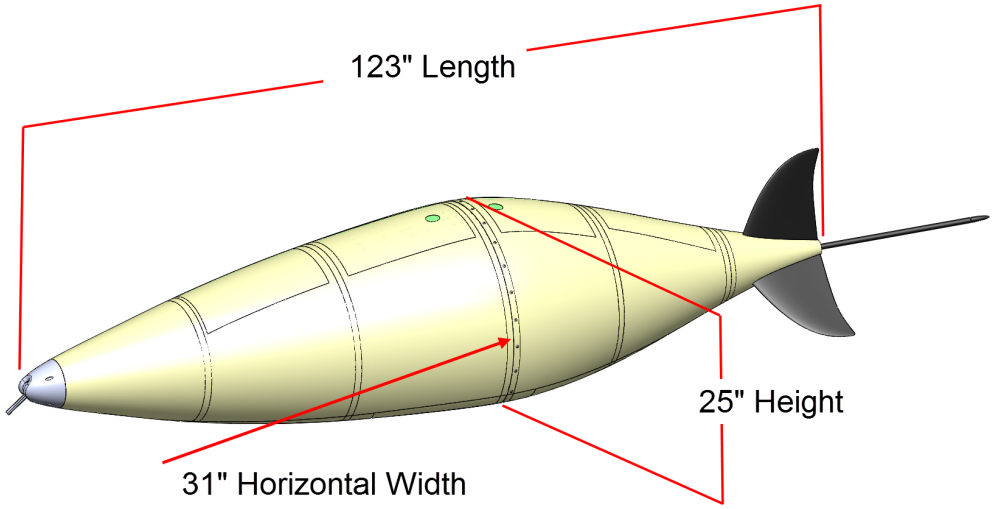


Example: Custom Large Payload Towfish

A towfish concept for very large payloads is illustrated below. This concept has been successful in initial sea trials and is included as an example of what is possible. After the payload(s) are installed, the Towfish is adjusted to neutral buoyancy and level trim.

Liquid Robotics Proprietary and Confidential

Figure 2.31. Example of a Large Payload Towfish - Dimensions in Inches



Chapter 3. Electrical Specifications

This chapter provides information on CCU and payload physical connectivity, including all standard electrical connections, and the Wave Glider SV3 Adaptable Modular Power System (AMPS).

Physical Connectivity

Figure 3.1 illustrates a Wave Glider system with several Float and Sub payloads connected. This is an example of what is possible; it should not be considered a limiting factor in the overall design and implementation of a Wave Glider system. The connections between Wave Glider components shown in the diagram are described in greater detail, including pin-outs, later in this chapter.

Figure 3.1. Wave Glider CCU Connection Diagram

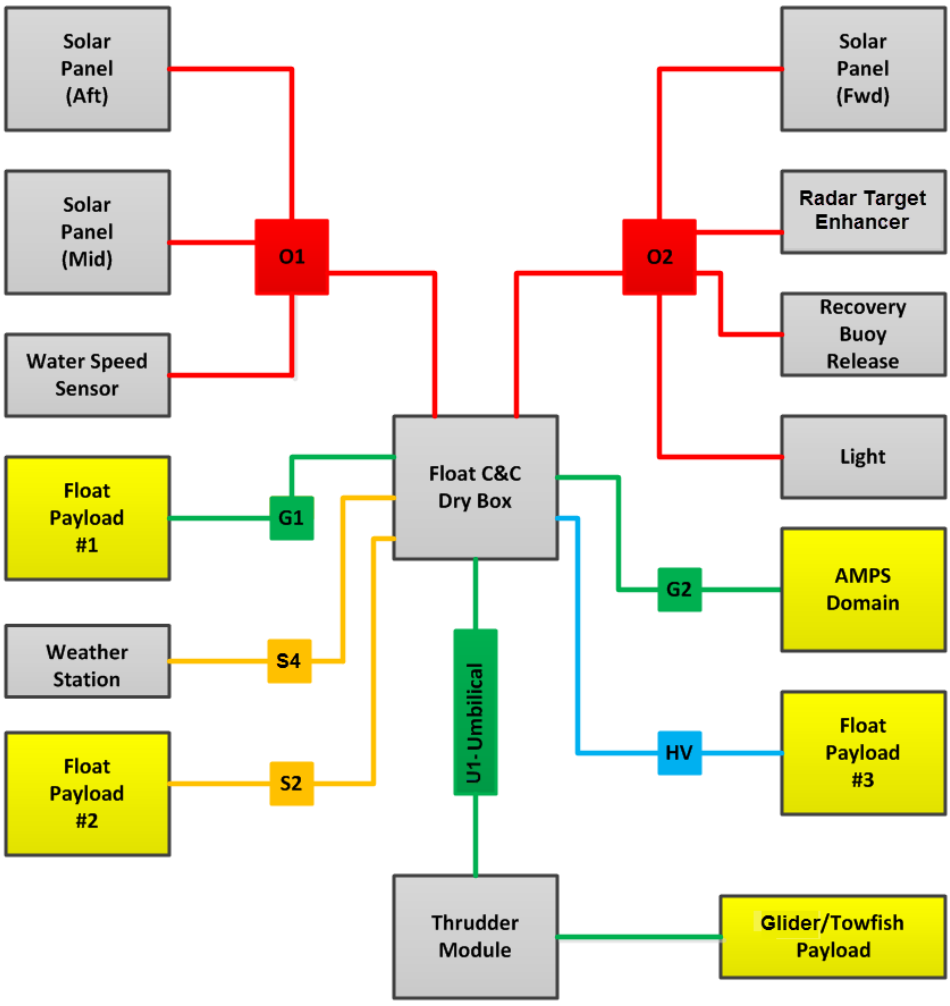


Table 3.1. Color and Label Conventions for Wave Glider Connection Diagram

Objects in Drawing	Description
Grey Boxes	These are standard Wave Glider system components.

Liquid Robotics Proprietary and Confidential

Objects in Drawing	Description
Yellow Boxes	These are payloads.
Green Lines	These are 12-pin General Expansion Port, Umbilical, and Thrudder Module Payload connections.
Tan Lines	These are 8-pin Sensor connections.
Blue Line	This is an 6-pin High Voltage Port connection.
Red Lines	These are 16-pin O1 and O2 Octopus Cable connections that branch out to a variety of standard Wave Glider components.

CCU Electrical

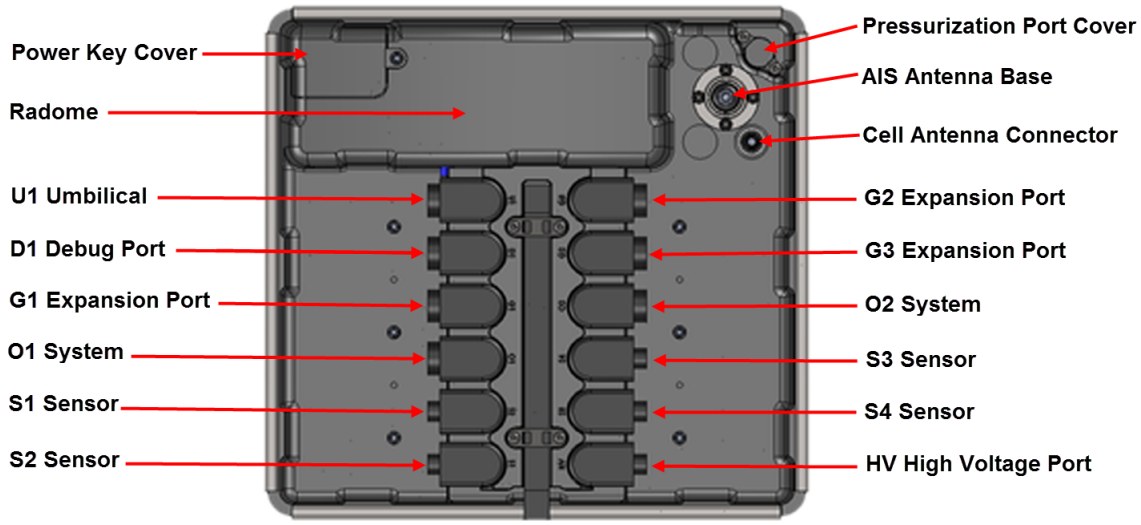
The CCU is the central hub for communications and power. This section describes the electrical interfaces to the CCU.

The CCU contains:

- Vehicle Management Computer (VMC)
- CCU AMPS Power Domain
- Iridium 9603 SBD Modem
- Primary GPS Module - UBLOX NEO-6Q
- Secondary GPS Module - integral to the CPU Module
- WiFi (2.4 & 5.0 GHz)
- Radome - Contains Iridium Antenna, Primary and Backup GPS Antennas, 2.4 GHz and 5.0 GHz WiFi Antennas.
- GSM Cell Modem
- Water Speed Sensor Interface Electronics
- Expansion Switch Matrix
- 8 Port Managed Ethernet Switch
- AIS Receiver
- Ten 98 watt hour Li-Ion Batteries
- 9 Axis Accelerometer
- Compass

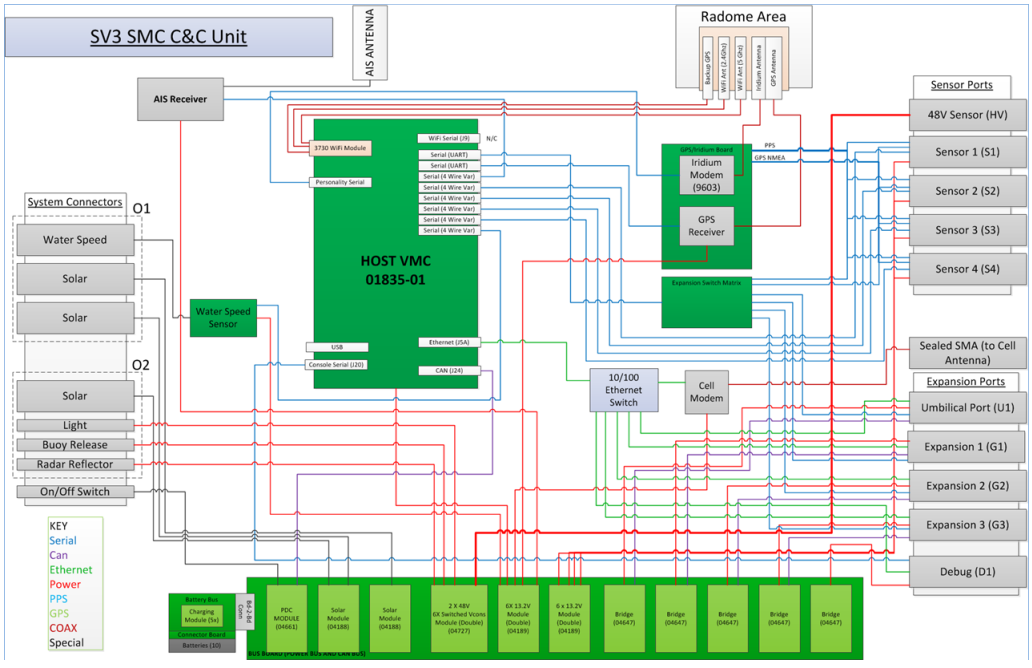
The CCU manages power for sensors and other payload components, and it provides communications paths for data retrieval and Wave Glider navigation. The CCU contains no customer-serviceable components.

Figure 3.2. CCU Electrical Connections



There are three types of CCU connectors of particular interest to System Integrators, General Purpose Expansion Ports (G1, G2, G3), Sensor Ports (S1, S2, S3, S4), and the High Voltage Port (HV). The Umbilical connector (U1) and the Debug Port (D1) are dedicated expansion ports.

Figure 3.3. CCU Electrical Block Diagram



For more information on the CCU, see “Command and Control Unit (CCU)” (page 8).

Liquid Robotics Proprietary and Confidential

G1, G2, G3 - General Purpose Expansion Ports

The 12-pin General Purpose Expansion Port connectors provide power, switched 10/100 Ethernet and communications channels to payloads and other AMPS domains. For more information see “G1, G2, G3 - General Purpose Expansion Port Connectors” (page 44).

S1, S2, S3, S4 - Sensor Connections

The 8-pin Sensor connections provide compatibility for Wave Glider SV2 payloads to be used on Wave Glider SV3 vehicles. For more information see “S1, S2, S3, S4 - Sensor Connectors” (page 47).

HV - High Voltage Connection

The 6-pin High Voltage connection provides up to 48V power for high voltage sensors, for example the ADCP, and it supports RS-232 communications with the CCU.

Note

The HV connection was not fully implemented on release 1.0 of the CCU.

For more information see “HV - High Voltage Connector” (page 50).

U1 - Umbilical

The 12-pin Umbilical connector is the power and communications connection between the Float and the Sub. The Umbilical transfers power from the Float to the Thrudder Module, and from there to Sub and Towfish payloads. The Umbilical also transfers data between the CCU and Float Payloads, Sub Payloads, the Thrudder Module and a Towfish. For more information see “U1 - Umbilical Connector” (page 45).

D1 - Debug Port

The 12-pin Debug Port is reserved for Wave Glider system troubleshooting and battery charging using an AC adapter. For more information see “D1 - Debug Port Connector” (page 46).

O1 - System Connection

The 16-pin O1 System connection uses a 16-pin "Octopus" cable to connect two solar panels and a water speed sensor to the CCU. For more information see “O1- System Connector” (page 53).

O2 - System Connection

The 16-pin O2 System connection uses a 16-pin "Octopus" cable to connect a solar panel, a radar target enhancer, a recovery buoy release mechanism, and a marker light to the CCU. For more information see “O2 - System Connector” (page 57).

Power Key

The power key turns the system on and off. It plugs into a 3 prong male electrical connector and is held in place with a threaded retaining nut. The system is powered on when the key is inserted. When

Liquid Robotics Proprietary and Confidential

the power key is removed, the system shuts down. When the system is shutdown, the batteries are individually disconnected and the solar panels and auxiliary power inputs are disconnected from the system.

Power Key Indicator Light

There is an LED indicator light in the handle of the power key.

- If the power key is inserted, but the light remains off, no power is available. If your batteries are charged and this condition persists, contact the Liquid Robotics Support Organization for assistance.
- If the power key is inserted, power is available, and no software updates are required, the light will begin blinking at a rate of 1 second on/1 second off for 10 seconds and then start blinking at a of 0.2 seconds on/2.8 seconds off. The light will continue to blink at this rate as long as the PDC for the CCU is running.
- If the power key is inserted, power is available, and a software update is required, the light will begin blinking at a rate of 1 second on/1 second off for 10 seconds and then start blinking at a higher variable rate during the software update. When the software update is completed the light will blink at a of 0.2 seconds on/2.8 seconds off, continuously, as long as the PDC for the CCU is running.
- If the power key is inserted and the light remains blinking at a rate of 1 second on/1 second off indefinitely, there is a problem. Contact the Liquid Robotics Support Organization for assistance.

Payload Connectivity

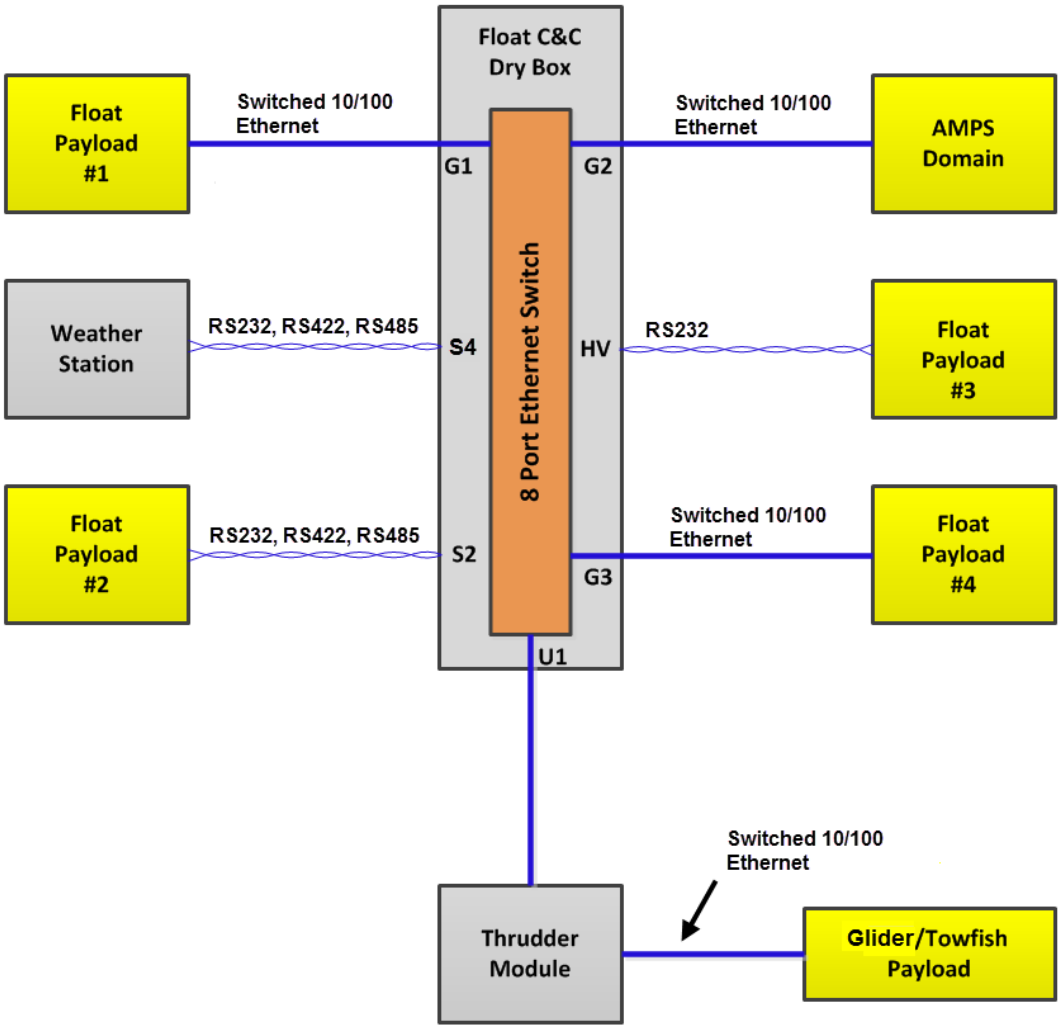
This section describes the physical connections between the Payload Dry Boxes and the Float CCU.

Payload Communication Connectivity

Figure 3.4 shows the communication paths available to Wave Glider payloads.

- Payloads connected to the CCU Sensor ports (S1, S2, S3, S4) can communicate directly with the CCU using RS232, RS422, or RS485 serial communications. Through the CCU, they can also communicate with other payloads, or with remote applications over the Iridium Network. Payloads that include a modem can communicate directly with remote applications.
- When the HV port is fully implemented, payloads connected to the CCU High Voltage port (HV) will be able to communicate directly with the CCU using RS232. Through the CCU, they will also be able to communicate with other payloads, or with remote applications over the Iridium Network. Payloads that include a modem will be able to communicate directly with remote applications.
- Payloads connected to the CCU General Purpose Expansion ports (G1, G2, G3) use switched 10/100 Ethernet communications. All communications sent or received by payloads connected to G ports pass through an Ethernet switch that can be configured to allow payload-to-payload communications or direct communications between payloads and the CCU. Payloads that include a modem can communicate directly with remote applications.
- Towfish or Sub payloads use switched 10/100 Ethernet communications available through the CCU Umbilical "U1" port (by way of the Thrudder module payload connector). All communications sent or received by Towfish or Sub payloads pass through the same Ethernet switch that is used by the "G" ports. The Ethernet switch can be configured to allow payload-to-payload communications or direct communications between payloads and the CCU.

Figure 3.4. Payload Communications Diagram



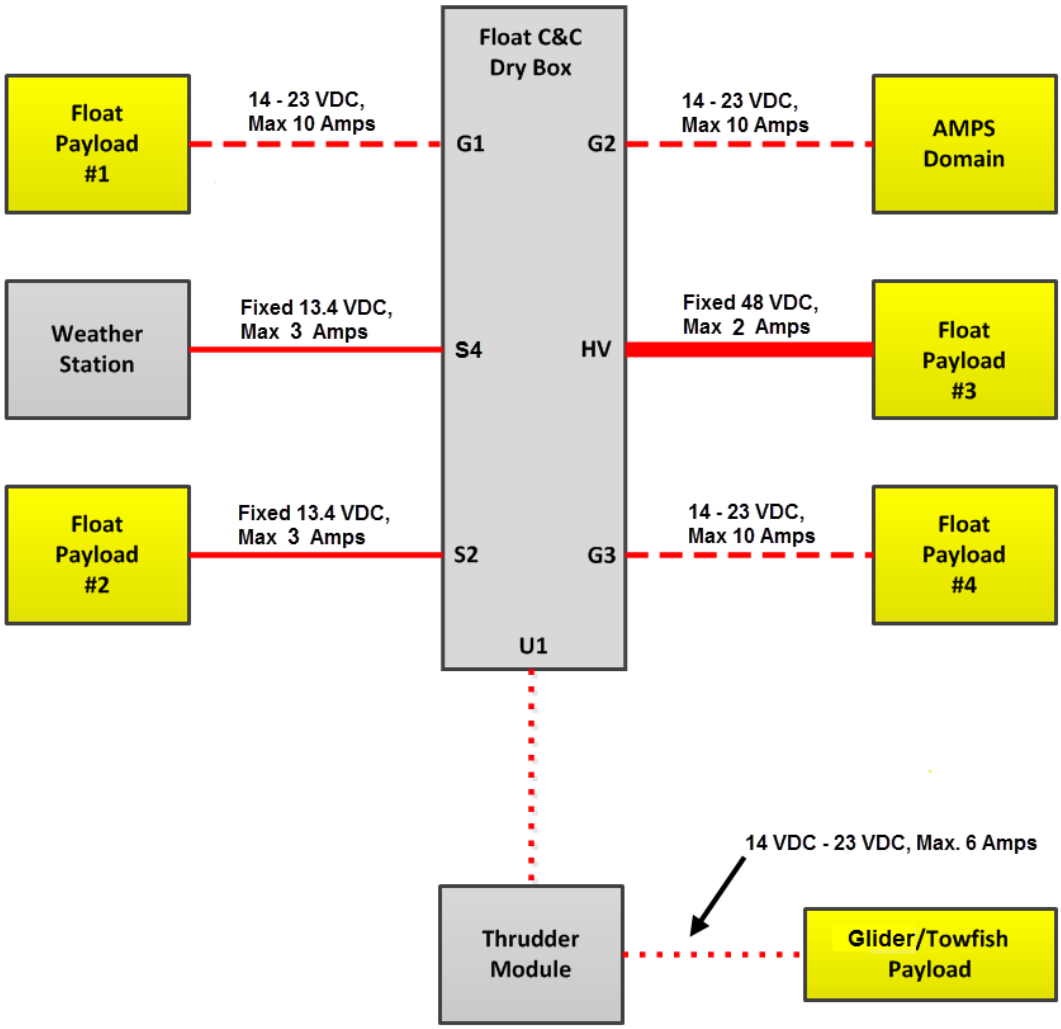
Payload Power Connectivity

Figure 3.5 shows power availability for Wave Glider payloads. Each of the payload connections to the CCU includes pins dedicated to powering a payload.

- Payloads connected to the CCU Sensor ports (S1, S2, S3, S4) receive a fixed 13.4 VDC power supply limited to a maximum of 3 Amps.
- Payloads connected to the CCU General Purpose Expansion ports (G1, G2, G3) receive a variable power supply ranging from 14 to 23 VDC limited to a maximum of 10 Amps.
- Payloads connected to the CCU High Voltage port (HV) receive a fixed 48 VDC power supply limited to a maximum of 2 Amps.
- Towfish or Sub payloads receive a variable power supply through the CCU Umbilical "U1" port (by way of the Thrudder module payload connector) ranging from 14 to 23 VDC and limited to a maximum of 6 Amps.

Liquid Robotics Proprietary and Confidential

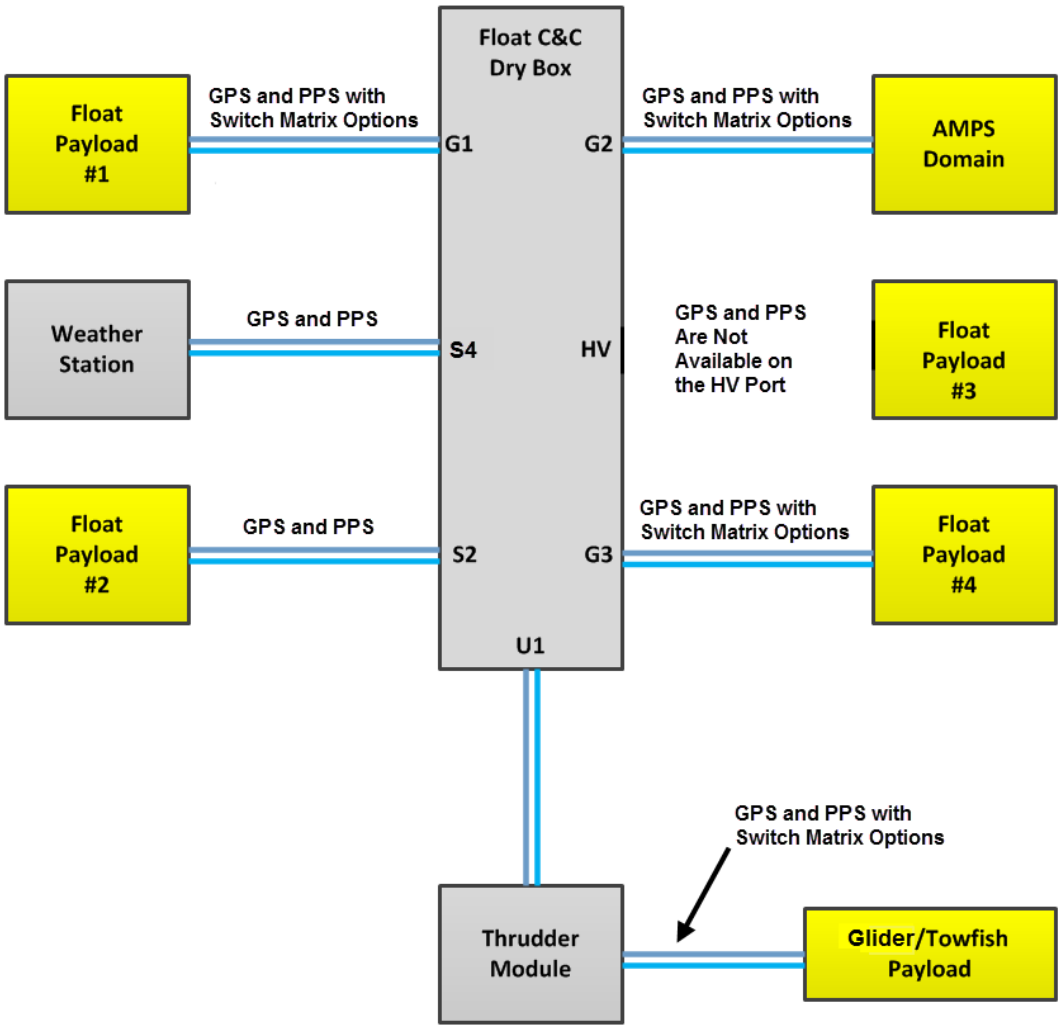
Figure 3.5. Payload Power Diagram



GPS and PPS Connectivity

Figure 3.6 shows GPS and PPS data availability for Wave Glider payloads. GPS and PPS data is provided through all of the CCU and Thrudder Module payload ports. For payloads connected to the G1, G2, G3, or Thrudder Module payload ports, a switch matrix makes GPS/PPS optional. If one of these payload connections is not configured (switched) to include GPS/PPS, the two lines in the connecting cable that support GPS/PPS can be used for other purposes. For more information on expansion port options see “G1, G2, G3 - General Purpose Expansion Port Connectors” (page 44) or “U1 - Umbilical Connector” (page 45).

Figure 3.6. GPS and PPS Diagram



Adaptable Modular Power System (AMPS)

The Adaptable Modular Power System (AMPS) enables extremely efficient power management for a wide range of possible Wave Glider configurations. By generating data on individual power producers and power consumers in prioritized power domains, AMPS facilitates optimal mission specific power management. AMPS is defined by a 3-wire power distribution system that includes hierarchical communications capability for monitoring and control.

AMPS extensibility supports:

- Scaling the number of power domains.
- Scaling power generation inputs.
- Scaling power storage capacity.

Liquid Robotics Proprietary and Confidential

AMPS scaling is achieved through the CCU Expansion Ports. The CCU Expansion Ports are Host Expansion Ports. A Host Expansion Port is defined by having a fully isolated AMPS interface that includes electronic isolation for VPRODS, VCONS, and CAN (AMPS power and communications).

A Guest Expansion Port is defined by having a partially isolated AMPS interface where VPROD and VCONS (AMPS power lines) are fully isolated, but the CAN interface (AMPS communication network) is not isolated and is always available.

AMPS is extended by connecting a Guest Expansion Port on an AMPS domain to:

- A Host expansion port on the CCU.
- A Host expansion port on an AMPS domain that is already extended from the CCU.

When a Guest Expansion Port on an AMPS domain, for example a payload or an Liquid Robotics Auxiliary Power Unit (APU), is connected to a Host Expansion port on the CCU or on another AMPS domain, a communication channel between the newly added AMPS domain and the CCU becomes active allowing you to configure and manage the newly added AMPS domain from a remote application (for example, WGMS). If you attempt to extend AMPS by connecting a Host Expansion Port on the newly added AMPS domain to a Host Expansion Port on the CCU (or on another AMPS domain), the newly added AMPS domain will remain isolated and non-operational. Any payload that includes an AMPS domain (AMPS backplane and modules) must include a Guest Expansion Port, and will not support further expansion of AMPS unless it also has a Host Expansion Port.

The AMPS configuration on a Wave Glider vehicle includes one or more power domains. Each power domain has a set of reusable, factory installed AMPS Modules that plug into the passive AMPS backplane, including a Power Domain Controller (PDC) module that is directed by the vehicle management computer in the CCU. There are AMPS modules for a variety of power management functions:

- “Power Domain Controller (PDC) Module ” (page 198)
- “Battery Charger Module” (page 201)
- “Solar Panel Input Module” (page 202)
- “High Voltage Module” (page 203)
- “Buck-Boost Regulated Output Module (BBR)” (page 206)
- “Bridge Module” (page 208)
- “1 MPU Module Carrier” (page 209)
- “3 MPU Module Carrier” (page 211)
- “Thrudder PDC” (page 213)
- “SMC Carrier” (page 215)

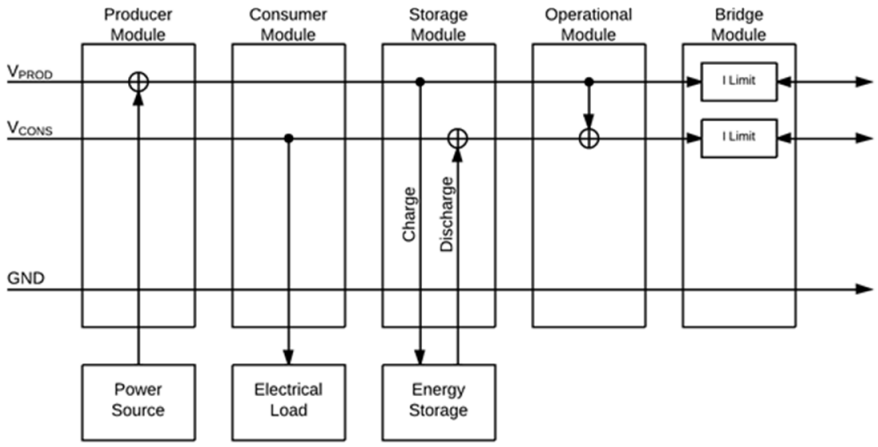
A Controller Area Network (CAN) supports communications between the CCU and the PDC in each AMPS domain. AMPS provides an intelligent interface between Wave Glider power producers and power consumers, switching them on or off to regulate power based on input to the CCU from local or remote applications.

AMPS also provides data buses and power buses:

Liquid Robotics Proprietary and Confidential

- AMPS data buses allow the PDC to send commands and receive power management data from all other AMPS modules in an AMPS domain.
- The AMPS power buses consists of a common ground circuit and two power circuits each capable of carrying up to 20A peak (240W at 12V). The two power circuits are referred to as Voltage Producer (VPROD) and Voltage Consumer (VCONS). VPROD receives all power input from the solar panels through the Solar Input modules then charges the batteries and feeds VCONS through the Battery Charger modules. VCONS receives power from the batteries through the Battery Charger modules. VCONS powers payloads through Power Consumer modules, and it also powers the VMC through the Host Power Supply module. VCONS can also feed VPROD through a Battery to Battery module.

Figure 3.7. AMPS Power Bus



Liquid Robotics Proprietary and Confidential

CCU Standard Electrical Connections

The Wave Glider SV3 CCU has 4 types of electrical connections:

- Five 12 Pin “CCU Expansion Ports” (page 41)
- Four 8 Pin “CCU Sensor Ports” (page 47)
- One 6 Pin “CCU High Voltage Connector” (page 49)
- Two 16 Pin “CCU System Connections” (page 50)

The following sections describe the connectors and pin-out for each connection to the CCU and for all connections to the Octopus cables.

CCU Expansion Ports

The 12 pin U1, D1, G1, G2, and G3 connections to the CCU are Host Expansion Ports. Expansion ports are the means for extending the SV3 power system (AMPS) and the means for providing high speed Ethernet communications to payloads. Expansion ports also provide a pair of programmable "SPARE" wires that can be used for communications or GPS services.

A Host Expansion Port is defined by having a fully isolated AMPS interface that includes electronic isolation for the VPRODS, VCONS, and CAN (AMPS power and communications lines).

A Guest Expansion Port is defined by having a partially isolated AMPS interface where the VPROD and VCONS (AMPS power lines) are fully isolated, but the CAN interface (AMPS communications) is not isolated and is always available.

AMPS is extended by connecting a Guest Expansion Port on an AMPS domain to:

- A Host Expansion Port on the CCU.
- A Host Expansion Port on an AMPS domain that is already extended from the CCU.

When a Guest Expansion Port on an AMPS domain, for example a payload or an Liquid Robotics Auxiliary Power Unit (APU), is connected to a Host Expansion port on the CCU or on another AMPS domain, a communication channel between the newly added AMPS domain and the CCU becomes active allowing you to configure and manage the newly added AMPS domain from a remote application (for example, WGMS). If you attempt to extend AMPS by connecting a Host Expansion Port on the newly added AMPS domain to a Host Expansion Port on the CCU (or on another AMPS domain), the newly added AMPS domain will remain isolated and non-operational. Any payload that includes an AMPS domain (AMPS backplane and modules) must include a Guest Expansion Port, and will not support further expansion of AMPS unless it also has a Host Expansion Port.

In addition to being Host Expansion Ports, the G1, G2 and G3 connections to the CCU are General Purpose Expansion Ports that are available for payloads or Auxiliary Power Units.

The U1 umbilical connection to the CCU is a Host Expansion Port dedicated to the Umbilical and the Thrudder Module Umbilical connection is a Guest Expansion Port. The Thrudder Module Payload Expansion Port is a Host Expansion Port that allows AMPS to expand to Sub or Towfish payloads. See “Thrudder Module Electrical Connections” (page 61) for more information.

The D1 Debug Port is a CCU Expansion Port, but the SPARE lines are hard wired to the Vehicle Management Computer (VMC) serial console port to facilitate debugging. The D1 connection to the

Liquid Robotics Proprietary and Confidential

CCU should be used for debugging and external battery charging only. For more information on external charging procedures using the Liquid Robotics Multi-Function Debug/Charging cable see "External Battery Charging" (page 68).

Table 3.2 provides information on each of the CCU Expansion Port pins listed in the Pin Out tables for the U1, D1, G1, G2, and G3 connectors.

Table 3.2. Expansion Port Signals

Signal Type	Description
GND and Shield	To prevent ground loops, the GND and Shield are grounded in one place. The shield is grounded on expansion ports with CAN signals originating from an AMPS bridge module.
VPROD	VPROD is the high current carrying bus that transports power from power producers, for example, a solar panel. VPROD can vary from 0 to 25 volts at 10 amps. NOTE: The Umbilical does not contain a wire for Pin 2, the VPROD, because there is no underwater power producer in the SV3 Wave Glider system.
VCONS	VCONS is the high current carrying bus that transports power to power consumers, for example, sensors. VCONS can vary from 14 to 25 volts at 10 amps (provided there is energy stored in the batteries).
CANH and CANL	These are the Controller Area Network (CAN) bus connections: • CAN bus H (AMPS Bridge CANH_PRI) • CAN bus L (AMPS Bridge CANL_PRI) They are terminated on the host system VMC, the Power Domain Controller (PDC), and Bridges. The CAN bus is not available for sensor integration, it is only for use between the VMC and the various AMPS domains.
TX+, TX-, RX+, RX-	These are the Ethernet connections. They terminate at an 8-port OEM Ethernet switch in the CCU (Sixnet ET-8MS-OEM): • 5 Ethernet switch ports are allocated to the U1, D1, G1, G2, and G3 Expansion Ports on the CCU. • 1 Ethernet switch port is allocated for the cellular network modem. • 1 Ethernet switch port is allocated to the VMC. • 1 Ethernet switch port is a spare.
SPARE1 and SPARE2	The SPARE signals for the U1, G1, G2, and G3 ports are programmable. By turning the expansion port relay on, SPARE signals can be connected to any other expansion port that has its relay turned on. Otherwise, SPARE1 is connected to GPS PPS and SPARE2 is connected to GPS DATA, similar to a sensor port (S1, S2, S3, S4). The signals are distributed at RS232 levels. SPARE1 and SPARE2 on the Debug Port connector (D1) connect to Vehicle Management Computer console in the CCU.

Liquid Robotics Proprietary and Confidential

Signal Type	Description

G1-G3 Options	Spare1	Spare2
Option 1	GPS NMEA	PPS
Option 2	VMC_RX*	VMC_TX*
Option 3	Umbilical Spare1	Umbilical Spare2
Option 4	No Connect	No Connect

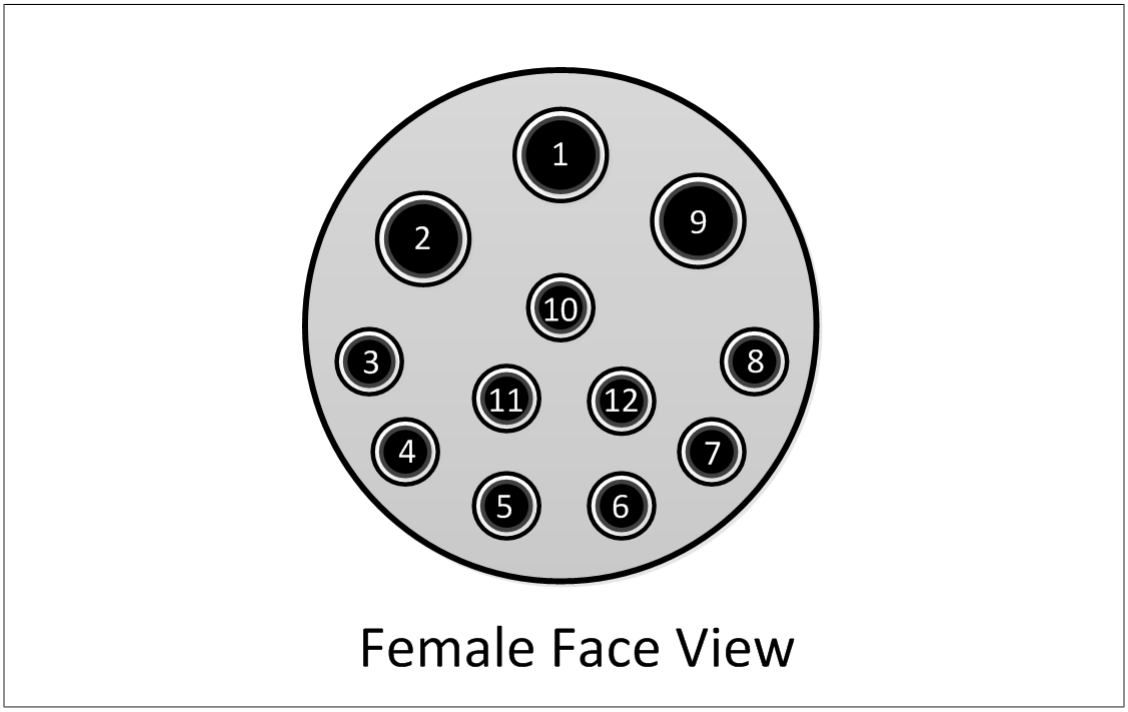
Umb Options	Spare1	Spare2
Option 1	GPS NMEA	PPS
Option 2	VMC_RX	VMC_TX
Option 3	G* Spare1	G* Spare2
Option 4	No Connect	No Connect

Debug Port	Spare1	Spare2
Only Option	VMC_console_RX	VMC_console_TX

* Not available at time of publication.

G1, G2, G3 - General Purpose Expansion Port Connectors

The G1, G2, and G3 General Purpose Expansion Port connections to the CCU support AMPS power management logic and also provide Ethernet switching capability for flexible payload-to-payload and payload-to-CCU communications management. They are designed to support payloads with multiple sensors or devices that require communications channels with other payloads and also with the CCU. This is in contrast to the S1, S2, S3, and S4 sensor connections to the CCU that are intended for stand-alone sensors running on a 13.4 volt power supply, with direct communications connections to the CCU.



General Purpose Expansion Port Pin Out

Pin	Signal	Description
1	GND	System ground plane
2	VPROD	AMPS Bridge VPROD
3	CANH	AMPS Bridge CANH_PRI
4	CANL	AMPS Bridge CANL_PRI
5	TX+	Ethernet Switch
6	TX–	Ethernet Switch
7	RX+	Ethernet Switch
8	RX–	Ethernet Switch
9	VCONS	AMPS Bridge VCONS
10	Shield	System ground plane
11	SPARE1	Programmable - See SPARE1 in Table 3.2.
12	SPARE2	Programmable - See SPARE2 in Table 3.2.

Note

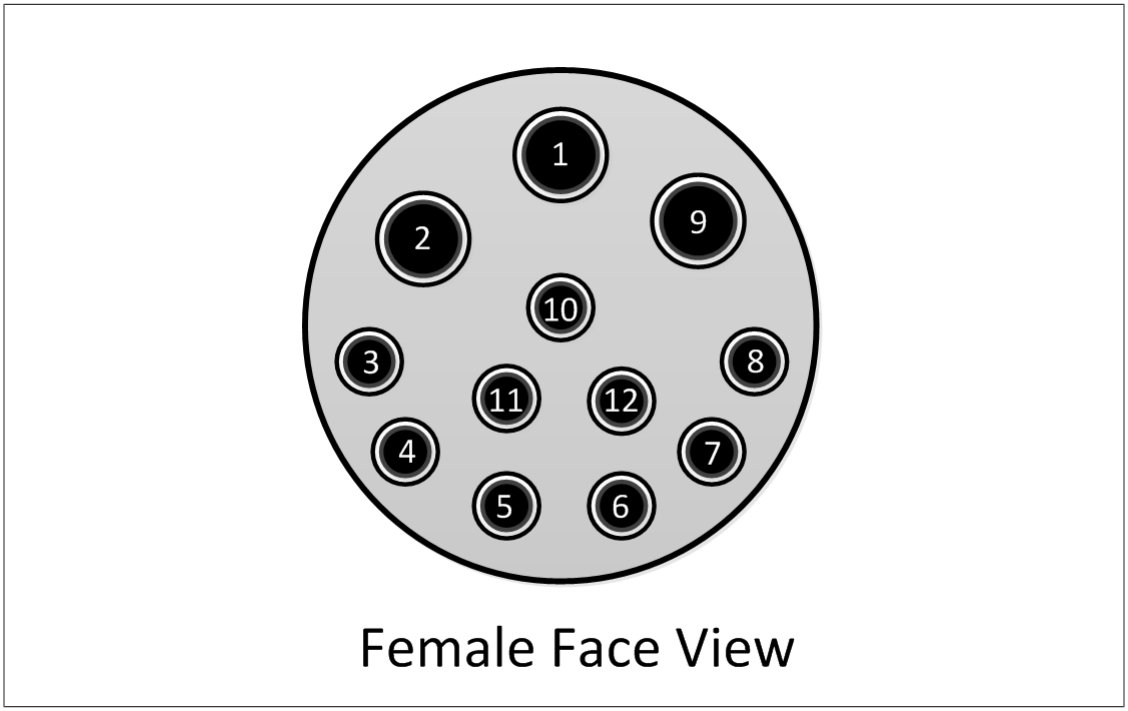
See Table 3.2 for additional information on each of the pins listed in the General Purpose Expansion Port Pin Out table.

General Purpose Expansion Port Mating Connector

The CCU General Purpose Expansion Port connectors mate with a wet mate-able 12 pin, male, whip assembly. These connectors are custom manufactured for Liquid Robotics by Teledyne.

U1 - Umbilical Connector

The Umbilical connection provides power and communications to the Sub and connectivity between the CCU, the Thrudder Module, and subsurface payloads. The Umbilical connection is a dedicated connector for the Umbilical. It is mechanically the same as a General Purpose Expansion Port, but pin 2 is not used because the Umbilical does not include a conductor for a high current power producer line (VPROD).



Umbilical Connector Pin Out

Pin	Signal	Description
1	GND	System ground plane
2	VPROD	AMPS Bridge VPROD
3	CANH	AMPS Bridge CANH_PRI
4	CANL	AMPS Bridge CANL_PRI
5	TX+	Ethernet Switch
6	TX–	Ethernet Switch
7	RX+	Ethernet Switch

Liquid Robotics Proprietary and Confidential

Pin	Signal	Description
8	RX–	Ethernet Switch
9	VCONS	AMPS Bridge VCONS
10	Shield	System ground plane
11	SPARE1	Interconnect Matrix
12	SPARE2	Interconnect Matrix

Note

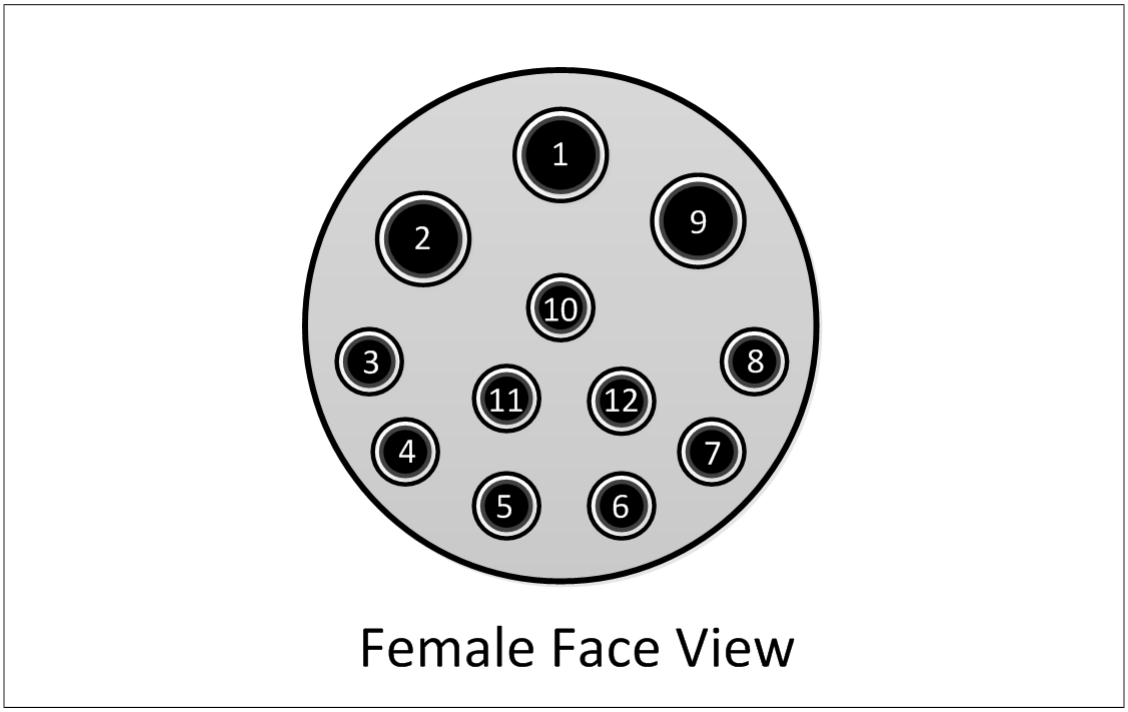
See Table 3.2 for additional information on each of the pins listed in the Umbilical Connector Pin Out table.

Umbilical Mating Connector

The CCU Umbilical connector mates with a wet mate-able, 12 pin, male whip assembly. These connectors are custom manufactured for Liquid Robotics by Teledyne.

D1 - Debug Port Connector

The Debug Port Connector is the primary maintenance port for the CCU. It is identical to a General Purpose sensor port except that the SPARE1 and SPARE2 lines are dedicated to providing access to the VMC's serial console port. This connector is also used for an external battery charger, although an external battery charger can be plugged into any CCU Expansion Port. For more information see “External Battery Charging” (page 68).



Debug Port Pin Out

Pin	Signal	Description
1	GND	System ground plane

Pin	Signal	Description
2	VPROD	AMPS Bridge VPROD
3	CANH	AMPS Bridge CANH_PRI
4	CANL	AMPS Bridge CANL_PRI
5	TX+	Ethernet Switch
6	TX–	Ethernet Switch
7	RX+	Ethernet Switch
8	RX–	Ethernet Switch
9	VCONS	AMPS Bridge VCONS
10	Shield	System ground plane
11	SPARE1	VMC Console
12	SPARE2	VMC Console

Note

See Table 3.2 for additional information on each of the pins listed in the Debug Port Pin Out table. The Umbilical and General Purpose Expansion Port connectors (U1, G1, G2, G3) connect to the interconnect matrix, but the SPARE1 and SPARE2 pins on the Debug Port connector (D1) connect to Vehicle Management Computer console in the CCU.

Debug Port Mating Connector

The CCU Debug Port connector mates with a wet mate-able 12 pin, male, whip assembly. These connectors are custom manufactured for Liquid Robotics by Teledyne.

CCU Sensor Ports

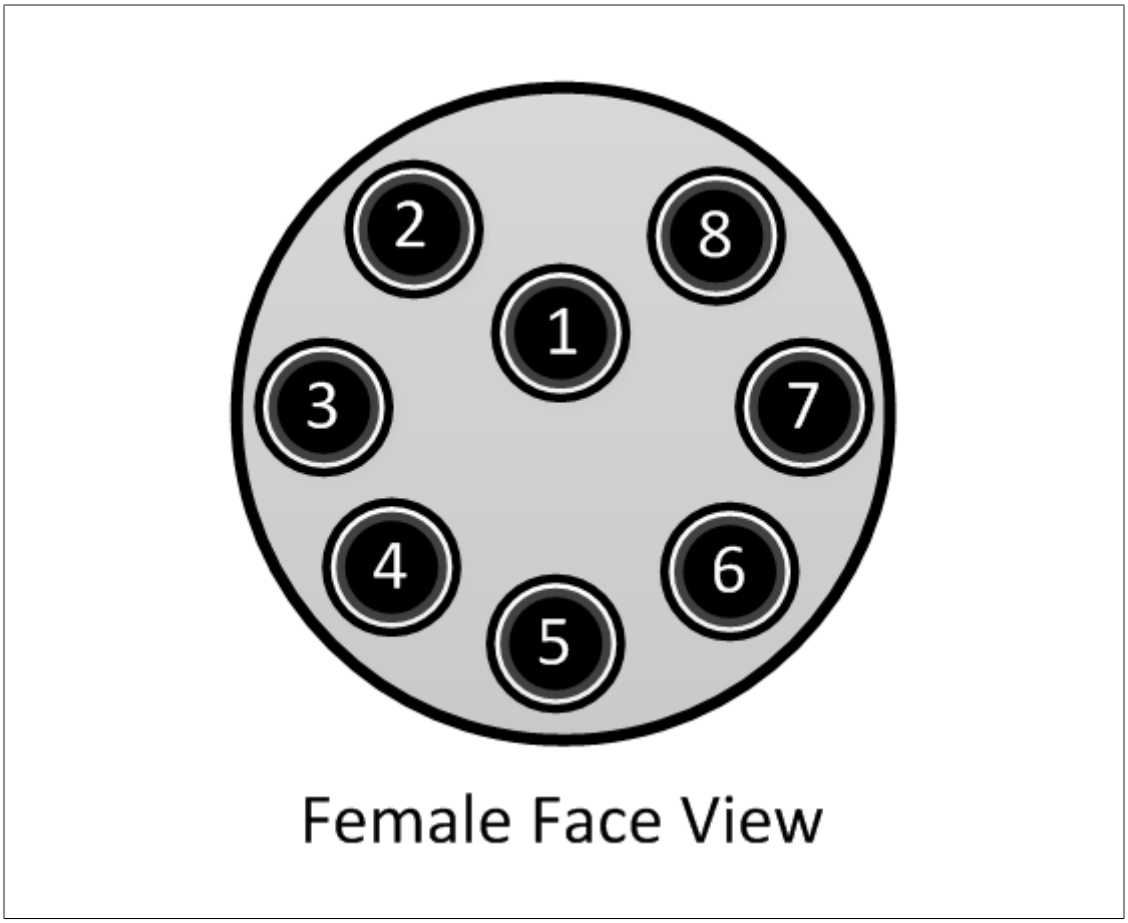
The 8 pin S1, S2, S3, and S4 Sensor Port connections to the CCU are intended for individual serially connected sensors. They can also be used to adapt Wave Glider SV2 payloads for use with the Wave Glider SV3 system. The default use for the CCU S4 Sensor Port is for the standard Wave Glider SV3 Weather Station.

S1, S2, S3, S4 - Sensor Connectors

The S1, S2, S3, and S4 sensor connections to the CCU are typically used for stand-alone sensors, or legacy Wave Glider SV2 payloads running on a 13.4 volt power supply. They support direct communications with the CCU using RS-232, RS-422, or RS-485, and they provide GPS and PPS at RS-232 levels.

Note

The SV3 "S" ports and the SV2 "Payload Ports" use the same 8 pin connectors, but they are NOT identical. The difference is that there are no AUX lines on the SV3 "S" ports. Instead the SV3 S ports support RS-232, RS-422, and RS-485. This is the same multi-protocol signal capability provided by the Liquid Robotics System Management Computer (SMC) that is typically used to manage Wave Glider integrated payloads. For more information see “Adapting Wave Glider SV2 Payloads for use on the Wave Glider SV3” (page 177).



S1, S2, S3, S4 Sensor Connector Pin Out

Pin	Signal	Description
1	PPS	VMC PPS
2	13.4V	AMPS BBR Output
3	TX	VMC UART/MAX3060E Driver
4	RX	VMC UART/MAX3060E Driver
5	RTS	VMC UART/MAX3060E Driver
6	CTS	VMC UART/MAX3060E Driver
7	GND	System Ground
8	GPS	VMC GPS

The following tables provide information on each of the S Port pins listed in the Pin Out table for the S1, S2, S3, and S4 connectors.

Table 3.3. Pins 1 and 8

Signal Type	Description
PPS and GPS	The PPS signal is a 100ms pulse - once per second, with positive polarity RS-232 pulses. The GPS data output is a 9600 baud, RS-232 NMEA format serial

Liquid Robotics Proprietary and Confidential

Signal Type	Description
	connection with GGA, ZDA, VTG, GLL, GST, and, RMC messages reported once every second. The order of the messages is not fixed.

Table 3.4. Pins 2 and 7

Signal Type	Description
13.4V Power and GND	The power pin provides a regulated 13.4 volts at up to 3.0A continuous draw. The port will handle surges of up to 3.5A. Any transient current draw greater than 3.5A for more than 160mswill constitute a fault and cause the output to shut off to protect itself. When sizing the load, care must be taken in sizing the total input capacitance seen by the power switch. If the capacitance is excessively large, the switch sees the output as being shorted, which causes a fault and the power will turn off to protect itself. The maximum allowable input capacitance for each payload port is ~50,000uF. If the equipment's transients are larger than 3.5A or the input capacitance is greater than 50,000uF, additional circuitry is required to buffer the power output. This can take the form of an NTC inrush limiter, local batteries, super capacitors, etc.

Table 3.5. Pins 3, 4, 5, and 6

Mode	Pin 3	Pin 4	Pin 5	Pin 6
RS-232	VMC_TX +	VMC_RX -	RTS	CTS
RS-422	VMC_TX -	VMC_RX +	VMC_TX +	VMC_RX -
RS-485	RS-485 -	N/A	RS-485 +	N/A

S1, S2, S3, S4 Sensor Mating Connector

The CCU S1, S2, S3, and S4 Sensor connectors are designed to mate to a wet mate-able connector from Teledyne or SubConn. The mating connector should be an 8 pin, male, whip type assembly. An example part number from SubConn is PNARA MCIL8M, which is the penetrator assembly used on the Liquid Robotics Forward and Aft Payload Dry Boxes for the Wave Glider SV2 Float.

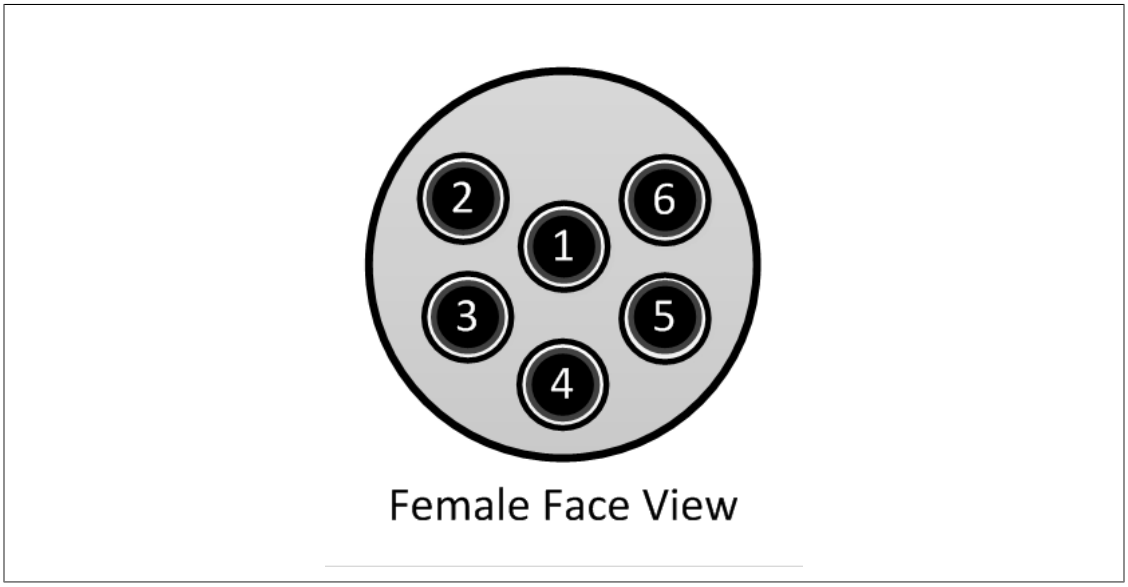
CCU High Voltage Connector

The 6-pin High Voltage connection to the CCU supports RS-232 communications and can provide up to 48 V of power.

Note

The HV connection was not fully implemented on release 1.0 of the CCU.

HV - High Voltage Connector



High Voltage Connector Pin Out

Cable Pin	Signal	Description
1	48V	AMPS HV Output - regulated 48V, limited at 2.0A
2	VMC_TX	VMC UART/MAX3060E Driver, RS-232 level
3	VMC_RX	VMC UART/MAX3060E Driver, RS-232 level
4	N/C	No Connection
5	N/C	No Connection
6	GND	Ground

High Voltage Mating Connector

The CCU High Voltage connector mates to a wet mate-able connector from Teledyne or SubConn. The mating connector must be a 6 pin, male, whip type assembly. An example part number from SubConn is MCIL6M, which is a whip assembly.

CCU System Connections

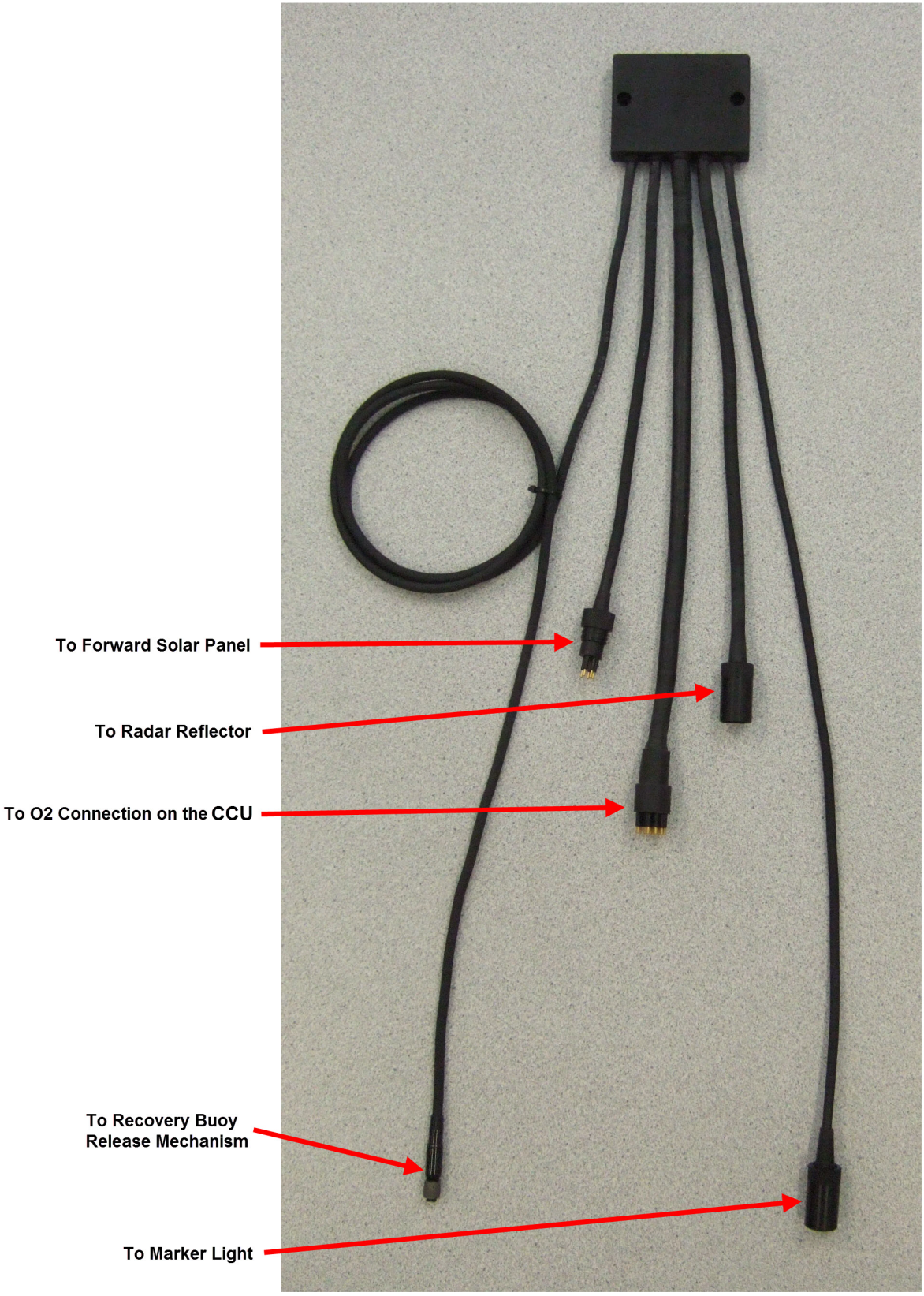
The CCU System Connections, O1 and O2, mate to two cable assemblies known as "Octopus Cables." Each Octopus Cable assembly includes a watertight junction box that mounts on the forward bulkhead of the aft payload bay on the Float. The Octopus Cables reduce wiring clutter and simplify the connections between the CCU and 7 standard Wave Glider SV3 devices.

Liquid Robotics Proprietary and Confidential

Figure 3.8. O1 Octopus Cable

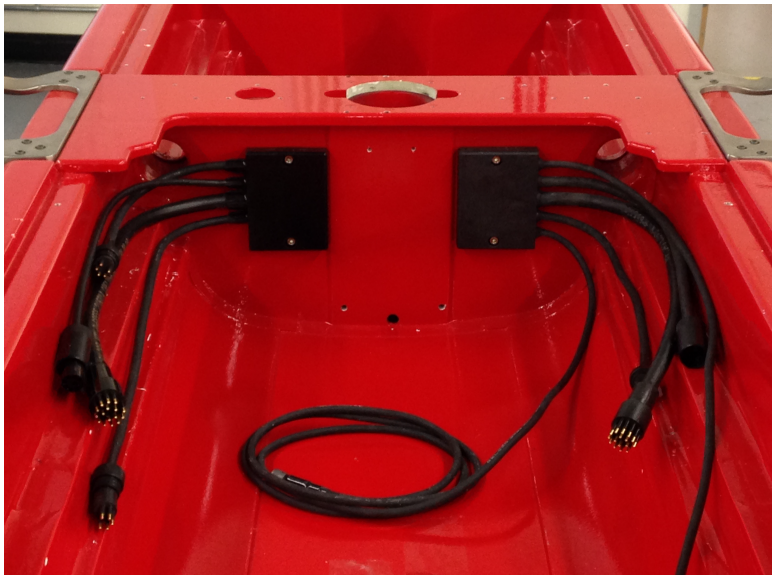
Liquid Robotics Proprietary and Confidential

Figure 3.9. O2 Octopus Cable



Liquid Robotics Proprietary and Confidential

Figure 3.10. Octopus Cable Assemblies Mounted to Bulkhead



There are four types of watertight connectors used on the Octopus cables.

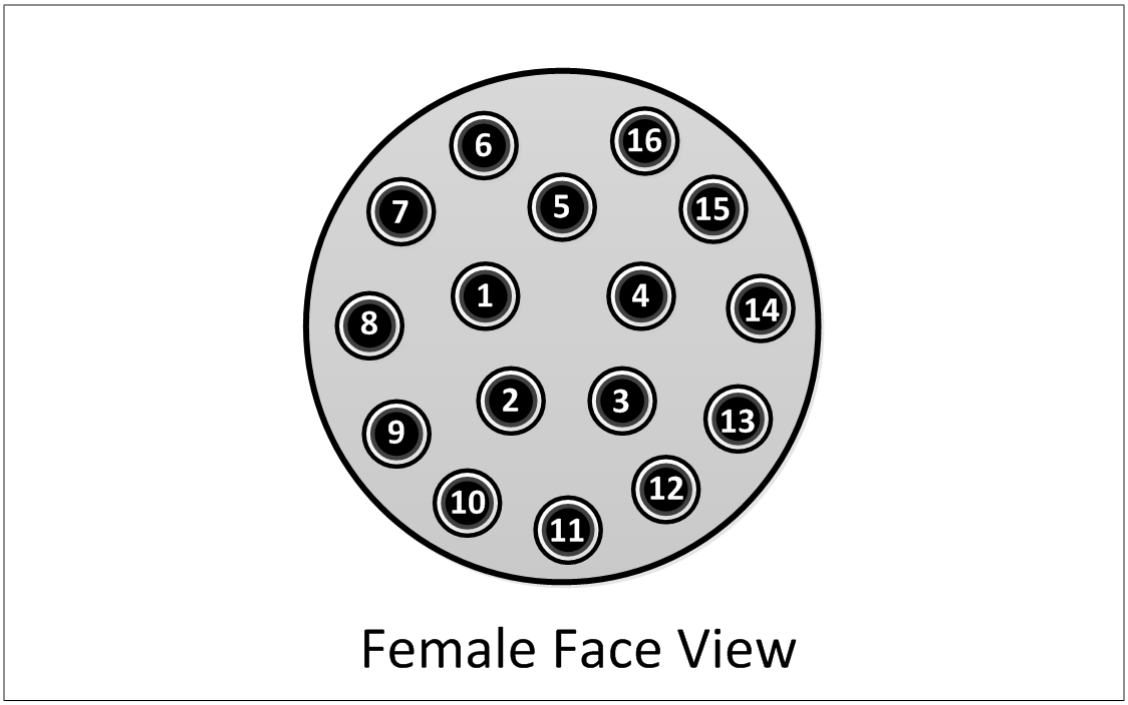
- 8-pin Water Speed and 8-pin Radar Target Enhancer connectors.
- 5-pin Solar Panel connectors.
- 3-pin Buoy Release connector.
- 2-pin Marker Light connector.

O1- System Connector

The O1-System connector is where the custom Liquid Robotics O1 Octopus cable connects to the CCU. The O1 Octopus cable includes an integral junction box that breaks-out to three smaller cables that connect to:

- Mid Solar Panel
- Aft Solar Panel
- Water Speed Sensor

Liquid Robotics Proprietary and Confidential

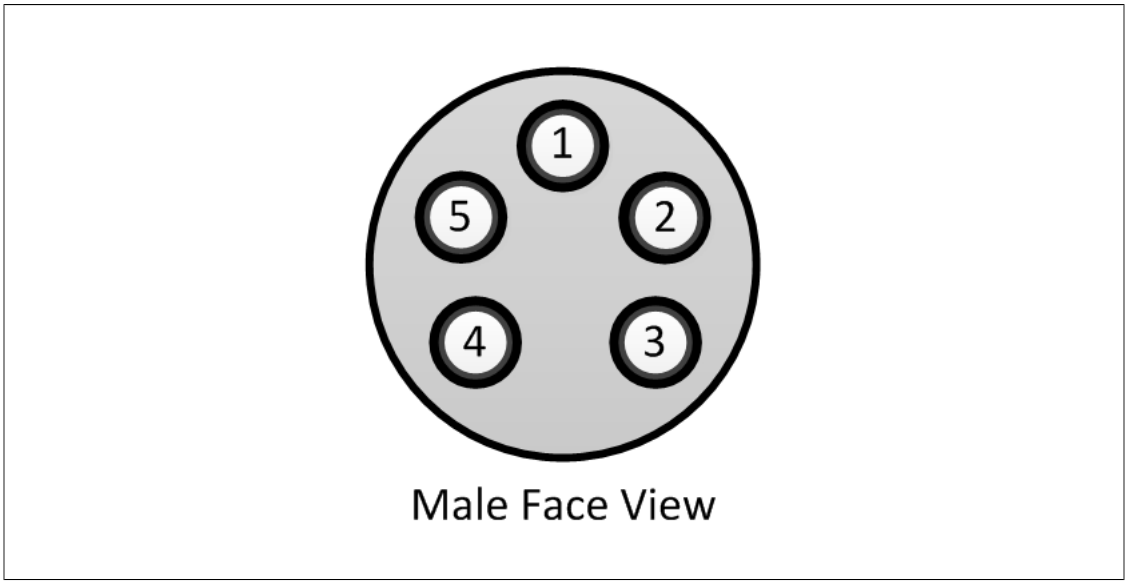


O1-System Connector Pin Out

Pin	Signal	Description
1	SOL1+	AMPS Solar Input
2	SOL1C	AMPS Solar Input
3	SOL1–	System Ground
4	SOL2+	AMPS Solar Input
5	SOL2C	AMPS Solar Input
6	SOL2–	System Ground
7	FORE_XD+	Fore Speed Sensor Module Input
8	FORE_XD-	Fore Speed Sensor Module Input
9	NC	NC
10	AFT_XD+	Aft Speed Sensor Module Input
11	AFT_XD-	Aft Speed Sensor Module Input
12	Shield	System Ground
13	TEMP+	Temperature Sensor Input
14	TEMP-	Temperature Sensor Input
15	KEYA	1k resistor between KEYA and KEYB
16	KEYB	

Liquid Robotics Proprietary and Confidential

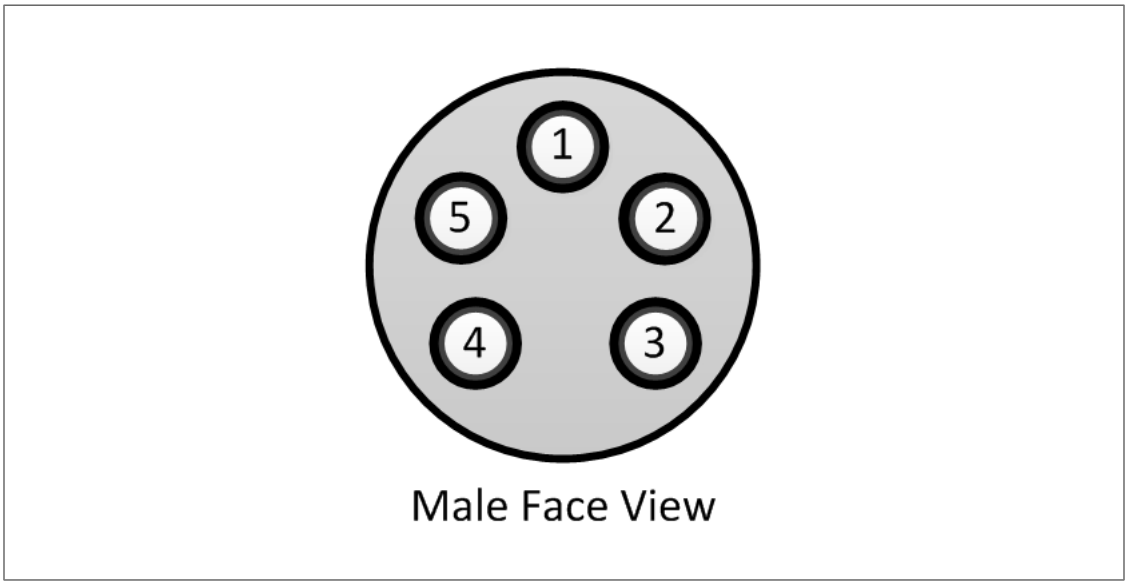
Mid Solar Panel Connector (Solar 1)



Mid Solar Panel Connector Pin Out

Pin	Connection #	Description
1	SYSCON1-1	SOLAR3_V+
2	SYSCON1-3	SOLAR3_V-
3	SYSCON1-2	SOLAR3_CT
4		N/C
5		N/C

Aft Solar Panel Connector (Solar 2)

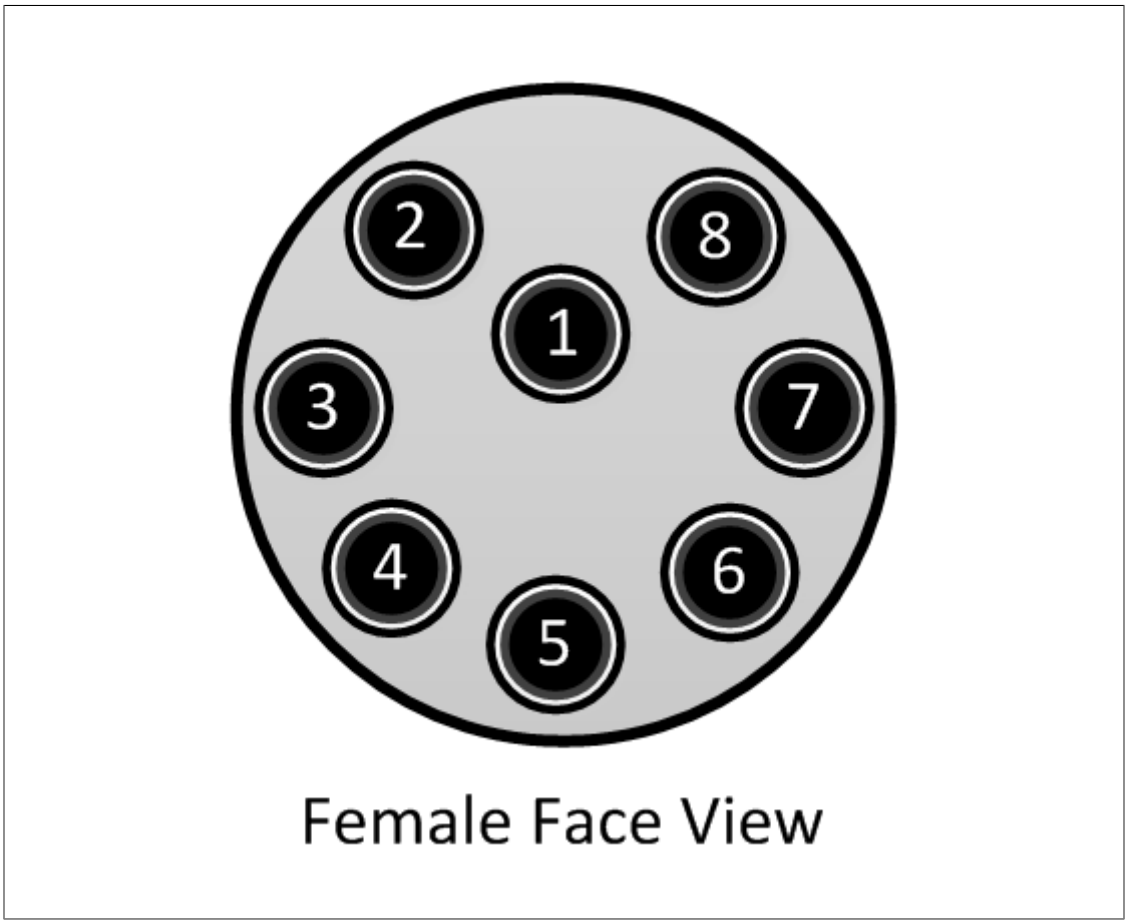


Liquid Robotics Proprietary and Confidential

Aft Solar Panel Connector Pin Out

Pin	Connection #	Description
1	SYSCON1-4	SOLAR3_V+
2	SYSCON1-6	SOLAR3_V-
3	SYSCON1-5	SOLAR3_CT
4		N/C
5		N/C

Water Speed Connector



Water Speed Connector Pin Out

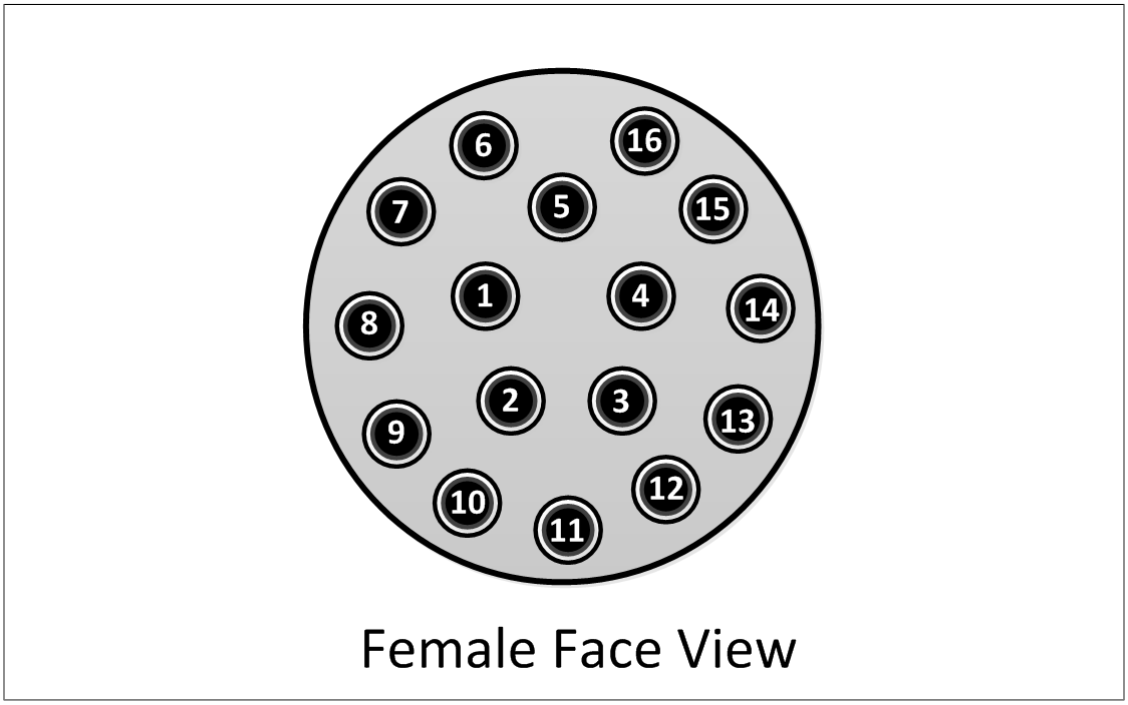
Pin	Connection #	Description
1	SYSCON1-13	TEMP+
2		N/C
3	SYSCON1-7	FORE_XD+
4	SYSCON1-8	FORE_XD-

Pin	Connection #	Description
5	SYSCON1-10	AFT_XD+
6	SYSCON1-11	AFT_XD-
7	SYSCON1-12	SHIELD
8	SYSCON1-14	TEMP-

O2 - System Connector

The O2-System connector is where the custom Liquid Robotics O2 Octopus cable connects to the CCU. The O2 Octopus cable includes an integral junction box that breaks-out to four smaller cables that connect to:

- Light
- Radar Target Enhancer
- Forward Solar Panel
- Buoy Release Mechanism



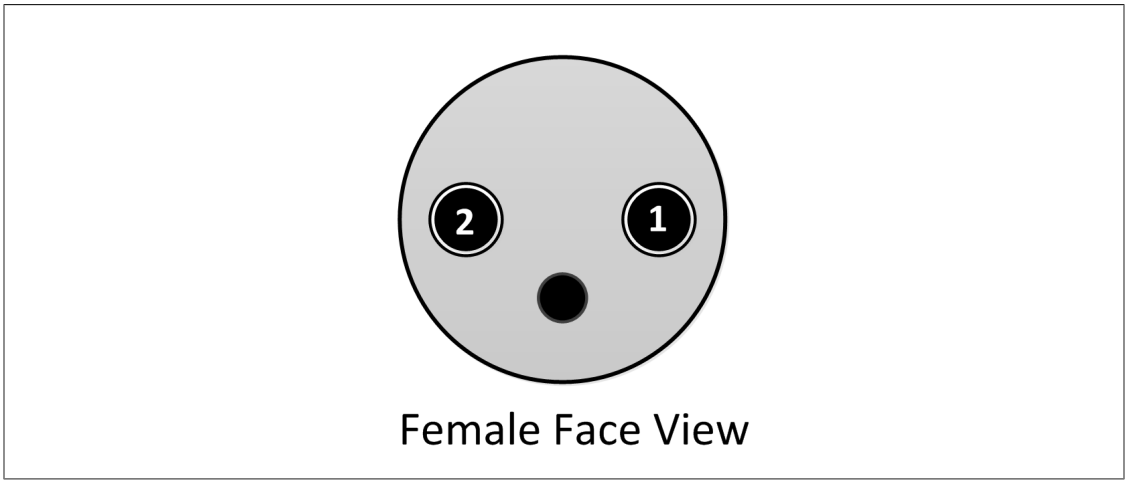
O2-System Connector Pin Out

Pin	Signal	Description
1	SOL3+	AMPS Solar Input
2	SOL3C	AMPS Solar Input
3	SOL3–	System Ground
4	SOL4+	AMPS Solar Input

Liquid Robotics Proprietary and Confidential

Pin	Signal	Description
5	SOL4C	AMPS Solar Input
6	SOL4–	System Ground
7	RADAR_PWR	AMPS HVP (unswitched)
8	RADAR_GND	System Ground
9	LIGHT_PWR	AMPS BBR Output
10	LIGHT_GND	System Ground
11	RECOVER_PWR	AMPS HVP (unswitched)
12	RECOVER_GND	System Ground
13	SYS_SPARE1	50 PIN ICB to BACKPLANE Connector
14	SYS_SPARE2	50 PIN ICB to BACKPLANE Connector
15	KEYA	2k resistor between KEYA and KEYB
16	KEYB	

Light Connector

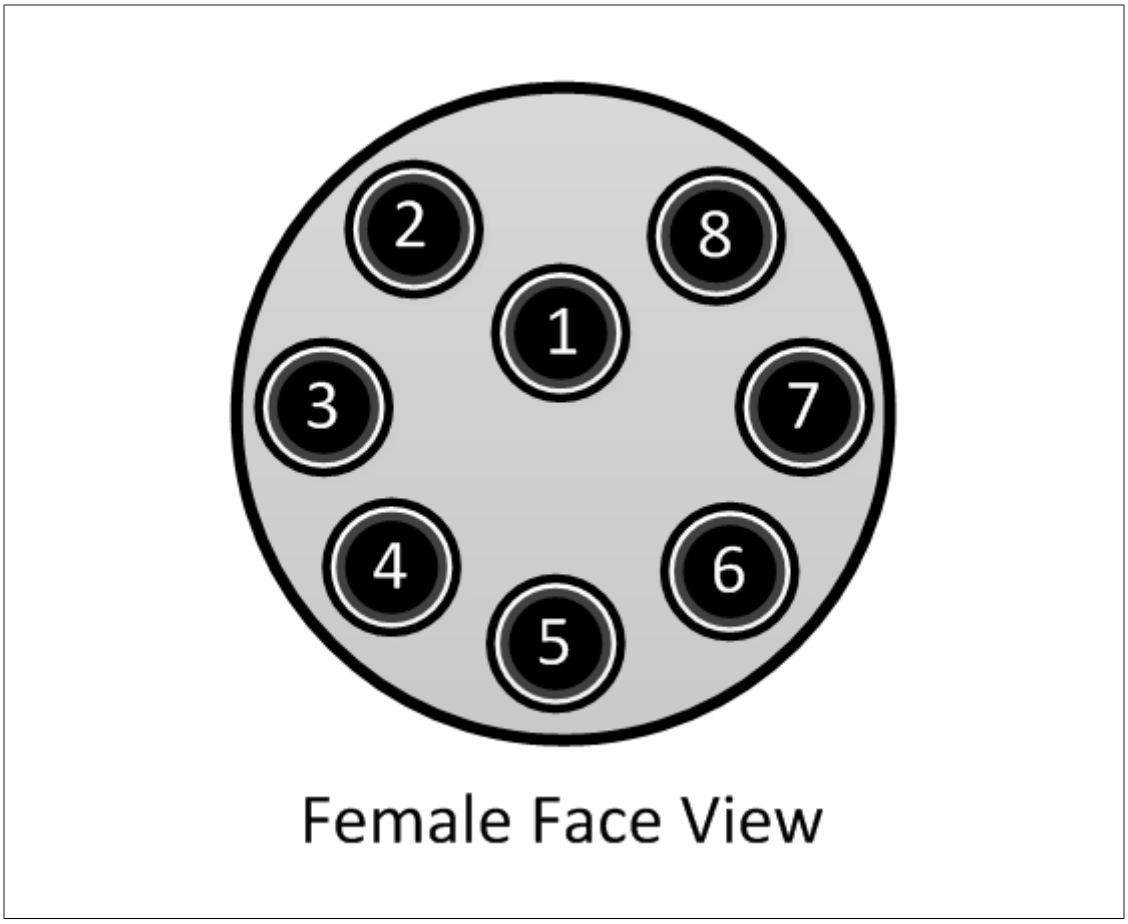


Light Connector Pin Out

Pin	Connection #	Description
1	SYSCON2-9	LIGHT_PWR
2	SYSCON2-10	LIGHT_GND

Radar Target Enhancer Connector

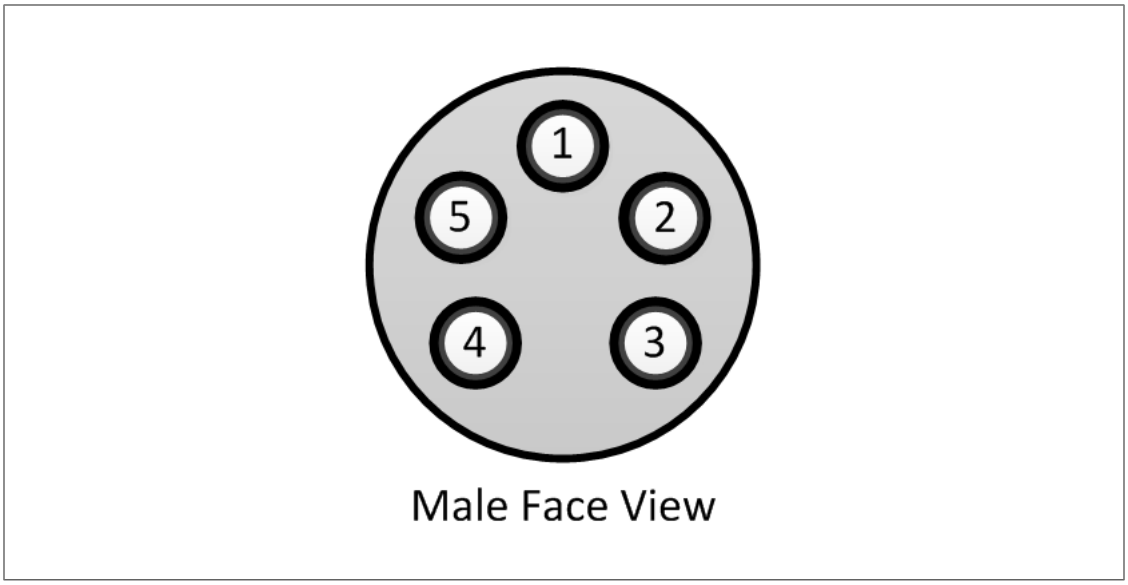
Liquid Robotics Proprietary and Confidential



Radar Target Enhancer Connector Pin Out

Pin	Connection #	Description
1	SYSCON1-13	TEMP+
2		N/C
3	SYSCON1-7	FORE_XD+
4	SYSCON1-8	FORE_XD-
5	SYSCON1-10	AFT_XD+
6	SYSCON1-11	AFT_XD-
7	SYSCON1-12	SHIELD
8	SYSCON1-14	TEMP-

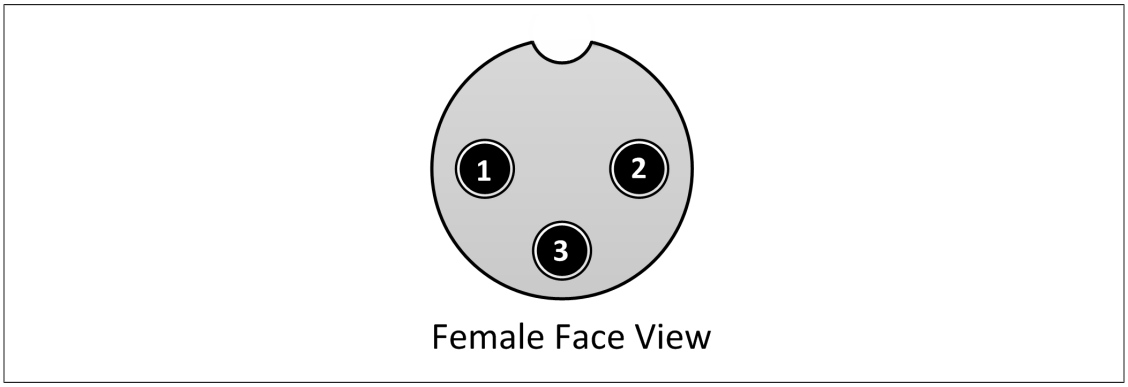
Forward Solar Panel Connector (Solar 3)



Forward Solar Panel Connector Pin Out

Pin	Connection #	Description
1	SYSCON2-1	SOLAR3_V+
2	SYSCON2-3	SOLAR3_V-
3	SYSCON2-2	SOLAR3_CT
4		N/C
5		N/C

Buoy Release Connector



Buoy Release Connector Pin Out

Pin	Connection #	Description
1	SYSCON2-11	RELEASE_PWR
2	SYSCON2-12	RELEASE_GND
3		N/C

Liquid Robotics Proprietary and Confidential

Thrudder Module Electrical Connections

Overview

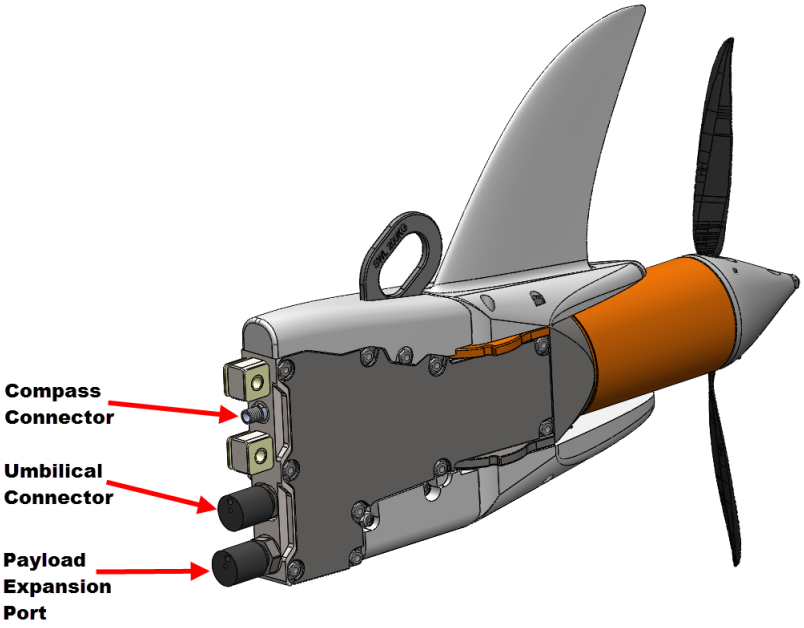
This section describes power and communications interfaces to the Wave Glider Thrudder Module. The Thrudder Module is the core component in compass based navigation, a control point for Sub and Towfish payloads, and a source of auxiliary propulsion when needed.

The Umbilical Connector interfaces through the Umbilical with the CCU.

The Payload Expansion Port is typically used to provide power and communications connectivity between Sub or Towfish payloads and the CCU.

The Compass Connector can support a compass mounted somewhere remote from the Thrudder Module on the Sub or on the Float. Liquid Robotics has not implemented this feature. The Compass Connector also provides a serial console debug connection to a processor that is housed in the Thrudder Module.

Figure 3.11. Thrudder Module Electrical Connections



Liquid Robotics Proprietary and Confidential

Figure 3.12. Thrudder Module Electrical Block Diagram

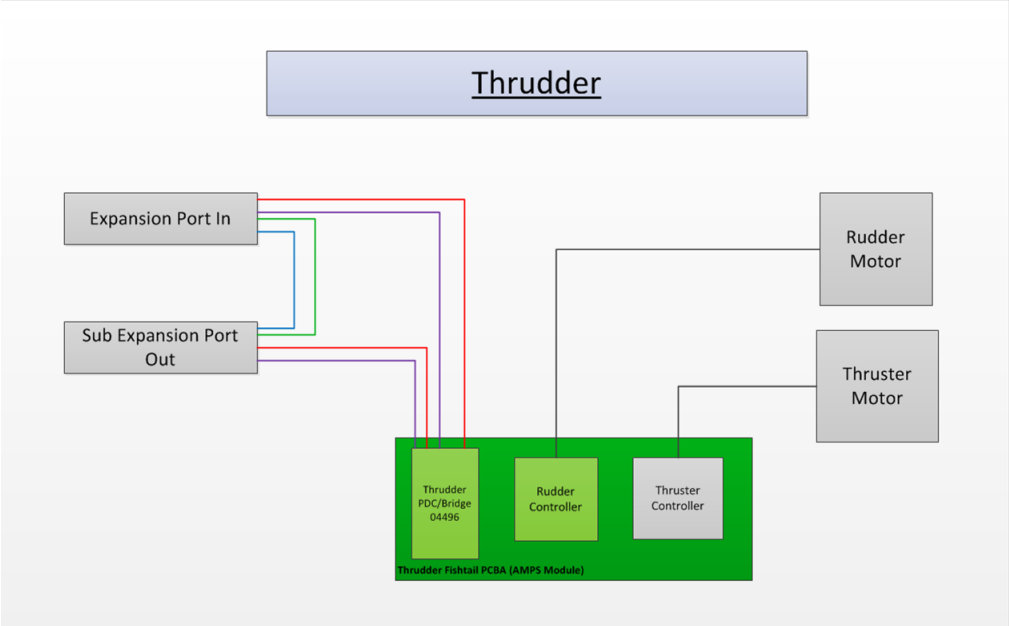


Table 3.6 provides information on each of the Thrudder Module Umbilical Connector and Expansion Port pins listed in the Pin Out tables for the Thrudder connectors.

Table 3.6. Thrudder Module Umbilical Connector and Expansion Port Signals

Signal Type	Description
GND and Shield	To prevent ground loops, the GND and Shield are grounded in one place. The shield is grounded on expansion ports with CAN signals originating from an AMPS bridge module.
VPROD	VPROD is the high current carrying bus that transports power from power producers, for example, a solar panel. VPROD can vary from 0 to 25 volts at 10 amps. NOTE: The Umbilical does not contain a wire for Pin 2, the VPROD, because there is no underwater power producer in the SV3 Wave Glider system
VCONS	VCONS is the high current carrying bus that transports power to power consumers, for example, sensors. VCONS can vary from 14 to 25 volts at 10 amps (provided there is energy stored in the batteries).
CANH and CANL	These are the Controller Area Network (CAN) bus connections: • CAN bus H (AMPS Bridge CANH_PRI) • CAN bus L (AMPS Bridge CANL_PRI) They are terminated on the host system VMC, the Power Domain Controller (PDC), and Bridges. The CAN bus is not available for sensor integration, it is only for use between the VMC and the various AMPS domains.

Liquid Robotics Proprietary and Confidential

<table>
<tr><th>Signal Type</th><th>Description</th></tr>
<tr><td>TX+, TX-, RX+, RX-</td><td>These are the Ethernet connections - they pass through the Thrudder Umbilical Connector to the Thrudder Payload Expansion Port connector. They terminate at an 8-port OEM Ethernet switch in the CCU (Sixnet ET-8MS-OEM):
5 Ethernet switch ports are allocated to the U1, D1, G1, G2, and G3 Expansion Ports on the CCU.
1 Ethernet switch port is allocated for the cellular network modem.
1 Ethernet switch port is allocated to the VMC.
1 Ethernet switch port is a spare.</td></tr>
<tr><td>SPARE1 and SPARE2</td><td>The SPARE signals for the Thrudder Module Umbilical connector and Expansion port are programmable - they pass through the Thrudder Umbilical Connector to the Thrudder Payload Expansion Port connector. By turning the expansion port relay on, SPARE signals can be connected to any other expansion port that has it's relay turned on. Otherwise, SPARE1 is connected to GPS PPS and SPARE2 is connected to GPS DATA, similar to a sensor port (S1, S2, S3, S4). The signals are distributed at RS232 levels. SPARE1 and SPARE2 on the Debug Port connector (D1) connect to Vehicle Management Computer console in the CCU.

<table>
<tr><th>G1-G3 Options</th><th>Spare1</th><th>Spare2</th></tr>
<tr><td>Option 1</td><td>GPS NMEA</td><td>PPS</td></tr>
<tr><td>Option 2</td><td>VMC_RX*</td><td>VMC_TX*</td></tr>
<tr><td>Option 3</td><td>Umbilical Spare1</td><td>Umbilical Spare2</td></tr>
<tr><td>Option 4</td><td>No Connect</td><td>No Connect</td></tr>
<tr><th>Payload Exp Port Options</th><th>Spare1</th><th>Spare2</th></tr>
<tr><td>Option 1</td><td>GPS NMEA</td><td>PPS</td></tr>
<tr><td>Option 2</td><td>VMC_RX</td><td>VMC_TX</td></tr>
<tr><td>Option 3</td><td>G* Spare1</td><td>G* Spare2</td></tr>
<tr><td>Option 4</td><td>No Connect</td><td>No Connect</td></tr>
<tr><th>Debug Port</th><th>Spare1</th><th>Spare2</th></tr>
<tr><td>Only Option</td><td>VMC_console_RX</td><td>VMC_console_TX</td></tr>
</table>

* Not available at time of publication.</td></tr>
</table>

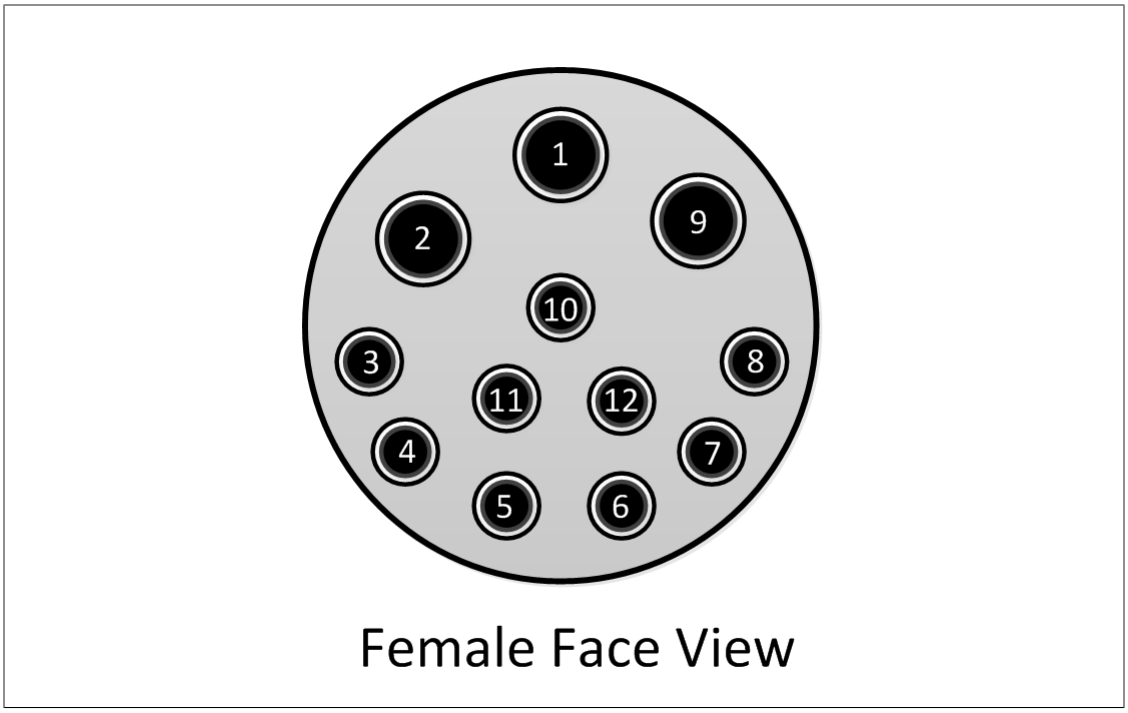
Thrudder Module Umbilical Connector

The Thrudder Module Umbilical connection provides power and communications to the Thrudder Module.

Note

The Umbilical does not contain a wire for Pin 2, the VPROD, because there is no power producer in the Sub.

Liquid Robotics Proprietary and Confidential



Thrudder Module Umbilical Connector Pin Out

Pin	Signal	Description
1	GND	System Ground Plane
2	This pin is not used and there is no conductor associated with it.	This pin is not used and there is no conductor associated with it.
3	CANH	AMPS Bridge CANH_PRI
4	CANL	AMPS Bridge CANL_PRI
5	TX+	Ethernet Switch
6	TX-	Ethernet Switch
7	RX+	Ethernet Switch
8	RX-	Ethernet Switch
9	VCONS	AMPS Bridge VCONS
10	Shield	System Ground Plane
11	SPARE1	Programmable - See SPARE1 in Table 3.6.
12	SPARE2	Programmable - See SPARE2 in Table 3.6.

Note

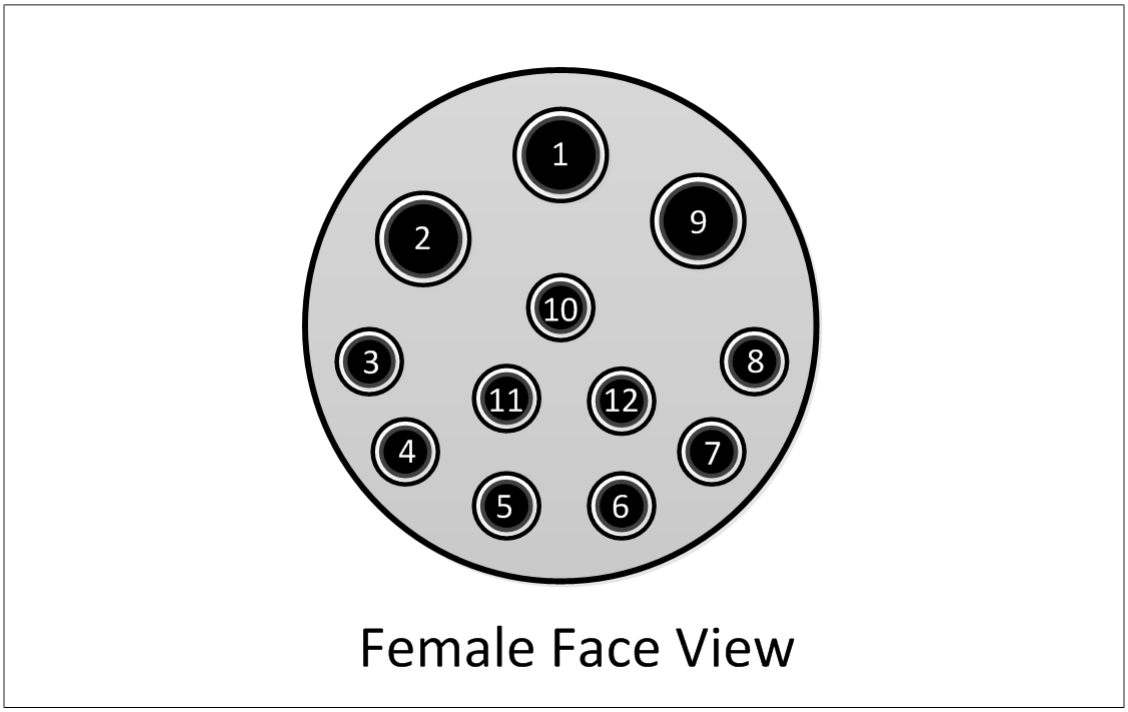
See Table 3.6 for additional information on each of the pins listed in the Thrudder Module Umbilical Connector Pin Out table.

Umbilical Mating Connector to the Thrudder Module

The Thrudder Module Umbilical port is designed to mate to the 12 pin male wet mate-able Thrudder electrical connector at the end of the Umbilical Cable Whip.

Thrudder Module Expansion Port Connector

The Thrudder Module Expansion Port connector provides power and communications connections to the CCU for Sub and Towfish payloads. The Thrudder Module Expansion Port connector is limited. There are no dedicated GPS, PPS or data path connections. If GPS, PPS or data path connections are required, they must be provided using the auxiliary lines.



Thrudder Module Expansion Port Pin Out

Pin	Signal	Description
1	GND	System ground plane
2	+12V	Regulated 12V @ 1A
3	CANH	AMPS Bridge CANH_PRI
4	CANL	AMPS Bridge CANL_PRI
5	TX+	Ethernet Switch
6	TX–	Ethernet Switch
7	RX+	Ethernet Switch
8	RX–	Ethernet Switch
9	VCONS	AMPS Bridge VCONS
10	Shield	System ground plane
11	SPARE1	Interconnect Matrix
12	SPARE2	Interconnect Matrix

Liquid Robotics Proprietary and Confidential

Note

See Table 3.6 for additional information on each of the pins listed in the Thrudder Module Expansion Port Pin Out table.

Thrudder Module Expansion Port Mating Connector

The Thrudder Module Expansion Port is designed to mate to a 12 pin male wet mate-able connector.

Unused Connections

Wave Gliders ship from the factory with "dummy plugs" installed on all unused connectors. Unused connections must have dummy plugs installed on them. If a connection is left open, it is susceptible to corrosion and leakage. Dummy plugs should also be installed when testing on a bench to protect the connectors from mechanical damage. If you need additional dummy plugs, make note of which connector(s) require dummy plugs and contact Liquid Robotics for ordering information. See “Contacting Liquid Robotics, Inc.” (page 264).

Batteries

The CCU contains a battery pack that supplies power to the core system and payloads as needed to supplement input from the solar panels. The CCU battery pack consists of ten 98 watt hour Lithium Ion batteries capable of storing up to 980 watt hours of energy. Optional Auxiliary Power Units (APUs) and stand-alone battery packs (for payload integration) are also available. An APU is a self-contained AMPS power domain. A stand-alone battery pack can be integrated into a payload power domain. Each APU or stand-alone battery pack also contains ten 98 watt hour Lithium Ion batteries capable of storing up to 980 watt hours of energy. AMPS allows energy sharing between the core electronics and the various payloads connected to the system.

Battery System Power

Each fully-charged Wave Glider CCU or APU Lithium-Ion battery pack has the following characteristics:

Battery Max Voltage	16.6 Volts
Maximum Continuous Current	20 Amps
Energy Capacity (fully charged) [theoretical maximum]	980 Watt-hours

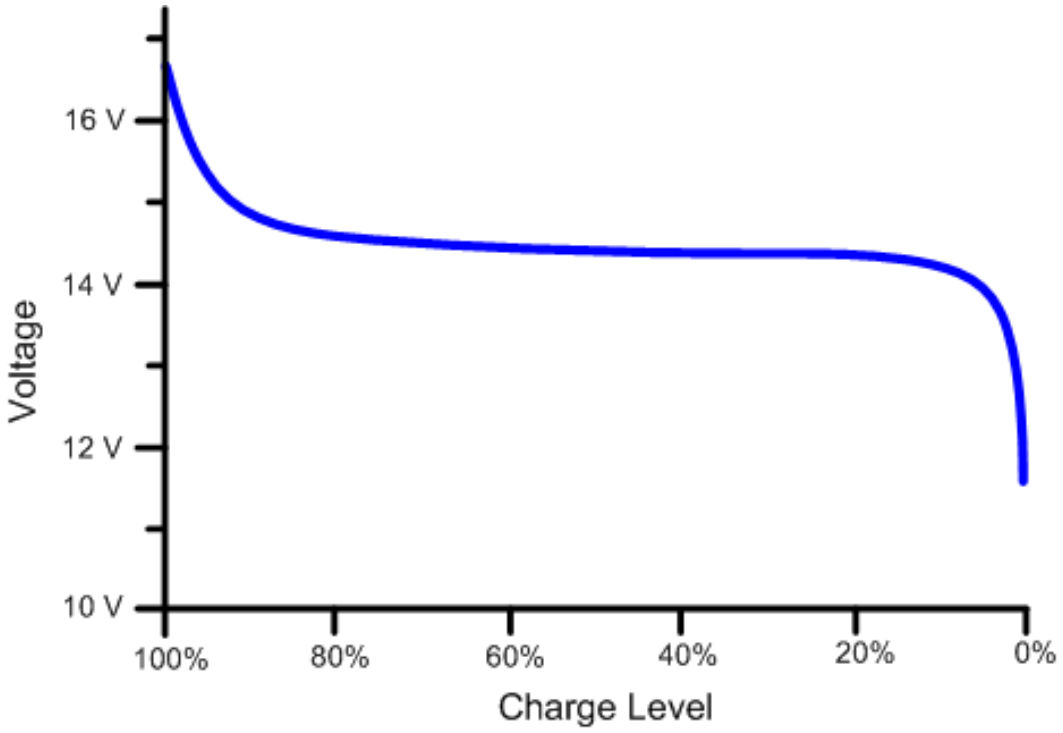
Note

The Maximum Continuous Current available to any payload depends on its connection to the system and if the payload has an integral battery pack. See the specifications for General Purpose Expansion Ports, Sensor Ports, and the High Voltage Port in “CCU Standard Electrical Connections” (page 41) for more information.

Battery Charge Level

The figure below shows how the battery voltage indicates the charge level. Note the characteristic rapid drop-off near full discharge.

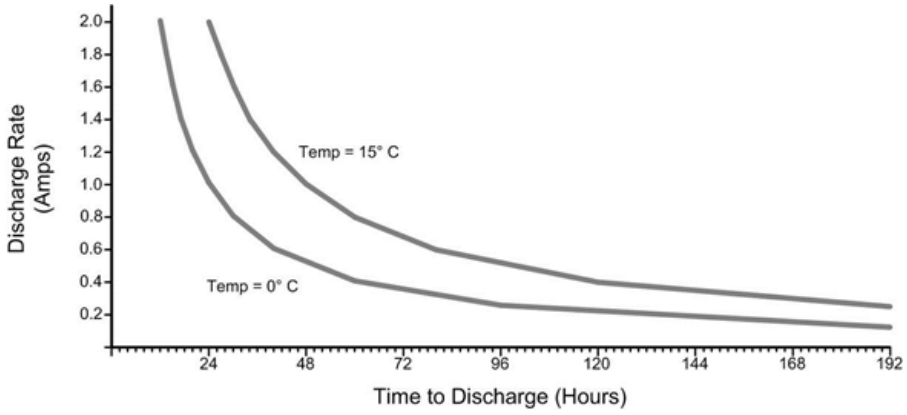
Figure 3.13. Battery Charge Level Relative to Voltage



Battery Discharge Characteristics

Assuming the batteries are fully charged, but there is no input from the solar panels, the discharge time will depend upon the Wave Glide current draw as follows:

Figure 3.14. Battery Discharge Time vs. Current Draw



Battery Power Input

The Float CCU is expecting a solar panel or similar type power source to provide energy to recharge the batteries. To minimize losses, the main path to charge the batteries is unregulated. For this reason, the power input to the batteries must fall within the ranges specified in “Solar Panel Specifications” (page 11).

There is also an auxiliary charging input available through the CCU Debug Port (D1). For more information see "External Battery Charging" (page 68).

Note

The Wave Glider batteries have a maximum charge acceptance temperature of 54C (129.2F). At temperatures above 54C the batteries will stop accepting a charge until they cool to less than 45C (113F). There is a tolerance of +/- 3deg C. For example, charging could stop at a temperature of 51C (123.8F) and charging could resume at 42C (107.6F).

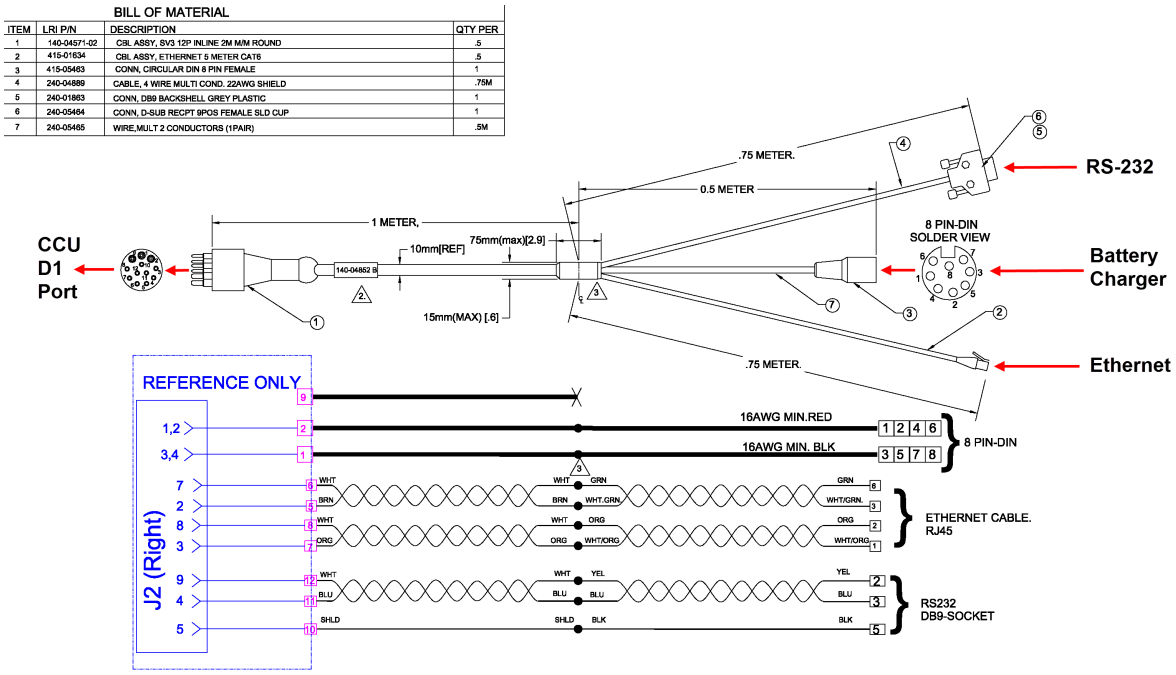
External Battery Charging

A multi-function cable and an external battery charger are included with each SV3 vehicle.

Liquid Robotics Multi-Function Debug/Charging Cable

The Liquid Robotics Multi-Function Debug/Charging Cable connects to the CCU Debug Port (D1) and is used for external battery charging and for debugging Wave Glider systems.

Figure 3.15. Liquid Robotics Multi-Function Debug/Charging Cable



Liquid Robotics Battery Charger

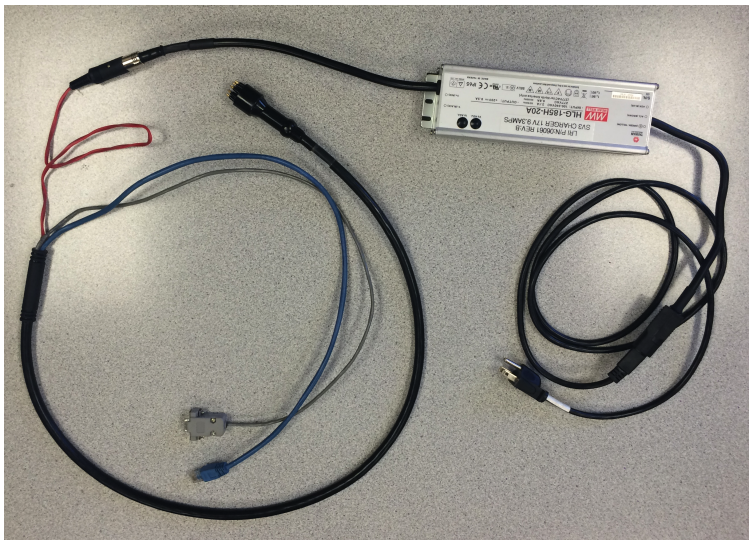
Table 3.7 provides the basic specifications for the external battery charger included with each Wave Glider vehicle. To avoid damage to the Wave Glider system, consult with Liquid Robotics, before attempting to substitute a different battery charger.

Liquid Robotics Proprietary and Confidential

Table 3.7. Liquid Robotics Battery Charger Specifications

Part Number	Output Voltage	Output Current	Peak Current (1)	Total Regulation (2)	Typical Efficiency (3)
Liquid Robotics P/N 05363	18.0V	5.6A	6.7A	±5%	89%
(1) Maximum peak load (120W) lasting 500ms with a maximum 10% duty cycle. (2) Includes initial setting, line regulation, load regulation, and thermal drift. (3) Typical at 115VAC (including output cable).					

Figure 3.16. Liquid Robotics Battery Charger



Charging the Batteries

To charge the batteries on a Wave Glider with a single Multi-Function Debug/Charging Cable and a single Battery Charger:

1. Connect the round 12 pin male connector on the Multi-Function Debug/Charging Cable to the D1 port on the Wave Glider CCU.

2. Connect the round 8 pin male connector on the battery charger to the round 8 pin female connector on the Multi-Function Debug/Charging Cable.

3. Connect the Battery Charger power line to a 120 VAC electrical outlet.

It does not matter if the Wave Glider system is powered-on, or powered-off, when you connect the Battery Charger, but the Power Key on the Wave Glider CCU must be inserted (powered-on) for battery charging to occur.

To charge the batteries on a Wave Glider with up to 4 Multi-Function Debug/Charging Cables and 4 Battery Chargers:

1. Connect the round 12 pin male connector on each Multi-Function Debug/Charging Cable to the D1, G1, G2, and G3 ports on the Wave Glider CCU.

2. Connect the round 8 pin male connector on each battery charger to a round 8 pin female connector on each Multi-Function Debug/Charging Cable.

3. Connect the Battery Charger power lines to 120 VAC electrical outlets.

It does not matter if the Wave Glider system is powered-on, or powered-off, when you connect the Battery Chargers, but the Power Key on the Wave Glider CCU must be inserted (powered-on) for battery charging to occur.

Warning

A maximum of 4 Liquid Robotics specified External Battery Chargers can be used simultaneously to charge the batteries on a single Wave Glider vehicle using Liquid Robotics Multi-Function Debug/Charging Cables.

APU Electrical Connections

Auxiliary Power Units (APUs) provide optional energy storage capacity. Adding an APU to a Wave Glider configuration doubles the power storage capacity of the Wave Glider system. Each APU includes a battery pack that consists of ten 98 watt hour Lithium Ion batteries capable of storing up to 980 watt hours of energy. This is the same battery pack that is included in the CCU.

Figure 3.17. Auxiliary Power Unit (APU) Electrical Connections

The 12 pin connections (A1 and A2) on each APU are referred to as APU Expansion Ports. The A2 connector is a Host Expansion Port and the A1 connector located next to the Power Key is a Guest Expansion Port. The differences between Host and Guest Expansion Ports are explained in “CCU Expansion Ports” (page 41).

APUs must connect to CCU General Purpose Expansion Ports. Two APUs can be daisy-chained together with the first APU in the chain connected directly to a CCU General Purpose Expansion Port. Alternatively, single APUs can be connected to different CCU General Purpose Expansion Ports.

Liquid Robotics Proprietary and Confidential

Each APU includes a 1 meter long connecting cable and a "dummy" plug. To connect a single APU to a CCU General Purpose Expansion Port, use the 1 meter cable to connect APU Expansion Port A1 (Guest Expansion Port) to a CCU General Purpose Expansion Port (Host Expansion Port), then connect a "dummy" plug to APU Expansion Port A2.

To daisy chain two APUs, use the 1 meter cable to connect APU Expansion Port A1 (Guest Expansion Port) to a CCU General Purpose Expansion Port (Host Expansion Port), then use another 1 meter cable to connect the A2 port of the first APU in the chain to the A1 port of the second APU in the chain. Connect a "dummy" plug to the A2 port of the second APU in the chain. It is important for APU connections to follow this pattern, otherwise they will not function.

Figure 3.18. APU Internal Battery Pack

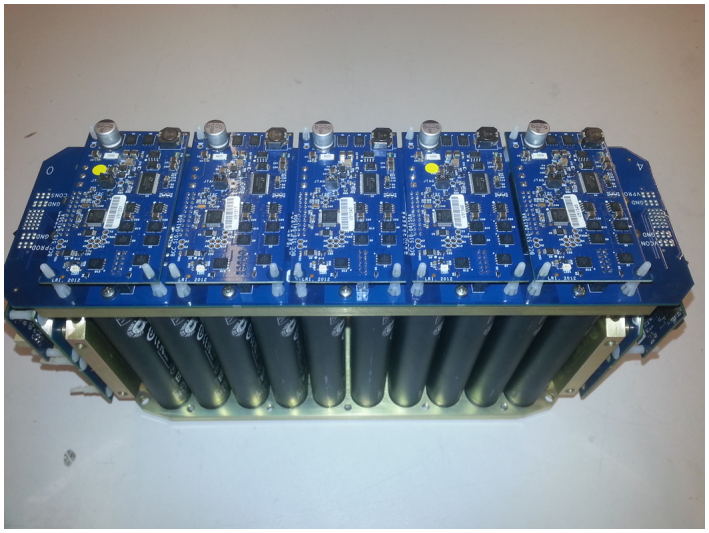
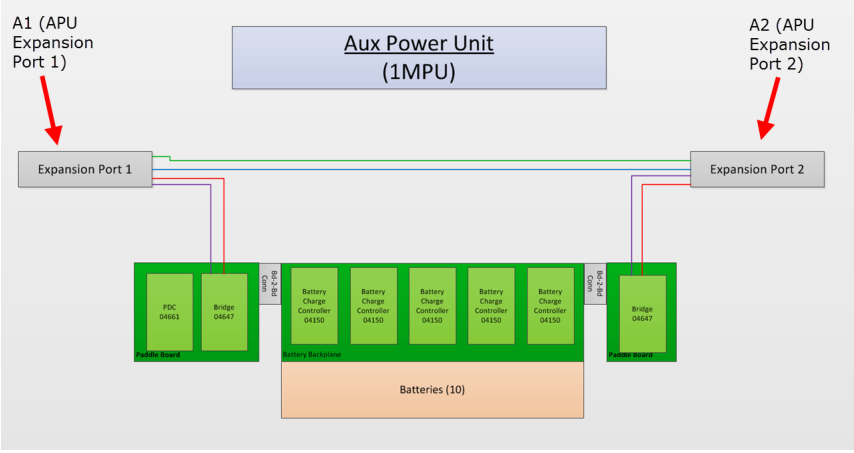


Figure 3.19. APU Electrical Diagram



Liquid Robotics Proprietary and Confidential

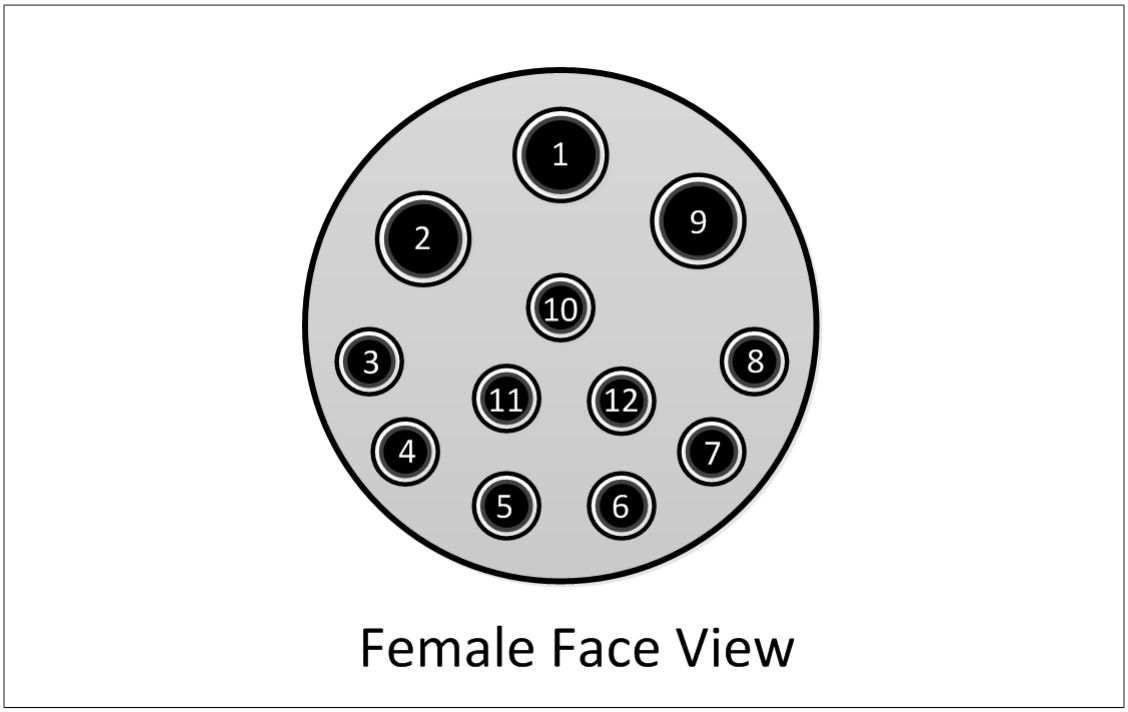
Table 3.8. APU Expansion Port Signals

Signal Type	Description
GND and Shield	To prevent ground loops, the GND and Shield are grounded in one place. The shield is grounded on expansion ports with CAN signals originating from an AMPS bridge module.
VPROD	VPROD is the high current carrying bus that transports power from power producers, for example, a solar panel. VPROD can vary from 0 to 25 volts at 10 amps. NOTE: The Umbilical does not contain a wire for Pin 2, the VPROD, because there is no underwater power producer in the SV3 Wave Glider system.
VCONS	VCONS is the high current carrying bus that transports power to power consumers, for example, sensors. VCONS can vary from 14 to 25 volts at 10 amps (provided there is energy stored in the batteries).
CANH and CANL	These are the Controller Area Network (CAN) bus connections: • CAN bus H (AMPS Bridge CANH_PRI) • CAN bus L (AMPS Bridge CANL_PRI) They are terminated on the host system VMC, the Power Domain Controller (PDC), and Bridges. The CAN bus is not available for sensor integration, it is only for use between the VMC and the various AMPS domains.
TX+, TX-, RX+, RX-	These are the Ethernet connections. They terminate at an 8-port OEM Ethernet switch in the CCU (Sixnet ET-8MS-OEM): • 1 Ethernet switch port in the CCU is allocated to the A2 Expansion Port on the APU when the A1 Expansion Port on that APU is connected to a Host Expansion Port.
SPARE1 and SPARE2	The SPARE signals for the APU A2 Expansion Port is connected by the interconnect matrix in the CCU when the APU A1 Expansion Port is connected to a Host Expansion Port. By turning the expansion port relay on, SPARE signals can be connected to any other expansion port that has it's relay turned on. Otherwise, SPARE1 is connected to GPS PPS and SPARE2 is connected to GPS DATA, similar to a sensor port (S1, S2, S3, S4). The signals are distributed at RS232 levels. SPARE1 and SPARE2 on the Debug Port connector (D1) connect to Vehicle Management Computer console in the CCU. A2 APU Expansion Port / Spare1 / Spare2 Option 1 / GPS NMEA / PPS Option 2 / VMC_RX* / VMC_TX* Option 3 / Umbilical Spare1 / Umbilical Spare2 Option 4 / No Connect / No Connect * Not available at time of publication.

Liquid Robotics Proprietary and Confidential

A1, A2 - APU Expansion Port Connectors

The A1and A2 APU Expansion Port connections support AMPS power management logic and also provide Ethernet switching capability for flexible payload-to-payload and payload-to-CCU communications management. A1 is a Guest expansion port and A2 is a Host expansion port.



APU Expansion Port Pin Out

Pin	Signal	Description
1	GND	System ground plane
2	VPROD	AMPS Bridge VPROD
3	CANH	AMPS Bridge CANH_PRI
4	CANL	AMPS Bridge CANL_PRI
5	TX+	Ethernet Switch
6	TX–	Ethernet Switch
7	RX+	Ethernet Switch
8	RX–	Ethernet Switch
9	VCONS	AMPS Bridge VCONS
10	Shield	System ground plane
11	SPARE1	Interconnect Matrix
12	SPARE2	Interconnect Matrix

Note

See Table 3.8 for additional information on each of the pins listed in the General Purpose Expansion Port Pin Out table.

Liquid Robotics Proprietary and Confidential

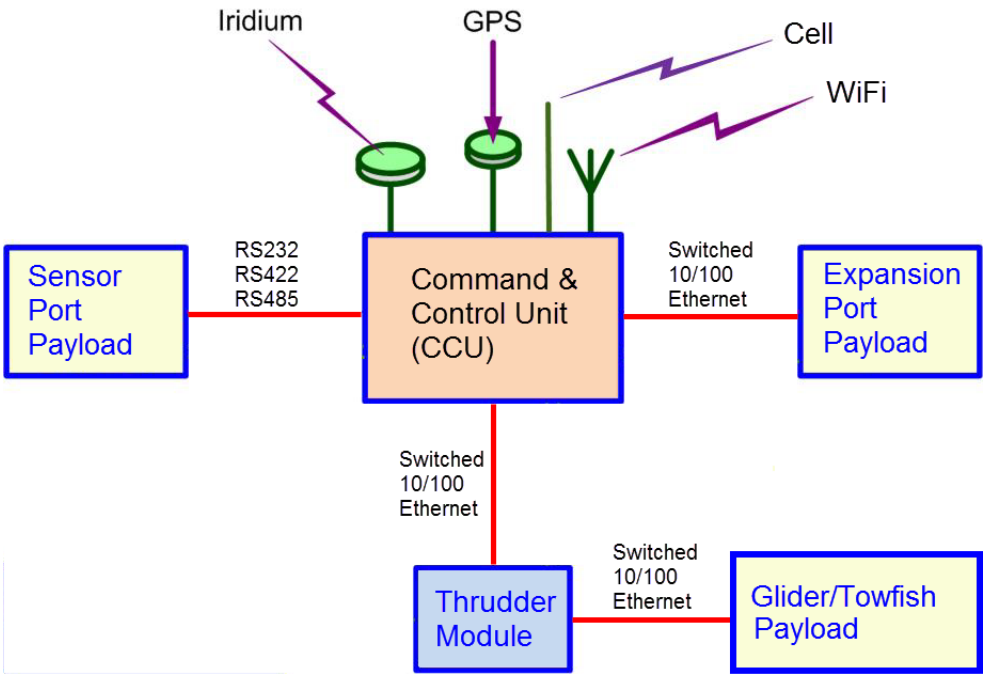
APU Expansion Port Mating Connectors

The APU Expansion Port connectors mate with the wet mate-able 12 pin connectors on the 1 meter cable assembly provided with each APU. To protect unused connector contacts, a "dummy" plug must be installed in the A2 connector on a stand-alone APU, or in the A2 connector on the last APU in a daisy-chain. A "dummy" plug is included with each APU.

Chapter 4. CCU Remote Communications and Control

Wave Glider vehicle command, telemetry and polling communications are managed by a computer contained in the Command & Control Unit (referred to as the "CCU"). The CCU handles communications between the Wave Glider and remote applications.

Figure 4.1. Wave Glider Remote Communications



This chapter includes information on:

- CCU Communication Functions
- CCU Communication Methods
- CCU Communication Data Types

CCU Communications Functions

This section provides an overview of the following CCU communications functions:

- Telemetry and Polling
- Command Processing
- Command and Message Responses

Telemetry and Polling

All communications with the CCU are initiated by the CCU. The CCU initiates communications when it transmits telemetry and polling to remote applications.

Liquid Robotics Proprietary and Confidential

The CCU automatically sends telemetry data over specified communications channels at regular, user-configurable, intervals. Telemetry data includes the position and general status of the vehicle, as well as any alarms. There are different telemetry data types, and each type can be sent out on a different schedule. For more information see Chapter 7: *Vehicle Telemetry*.

The CCU also sends periodic polling messages to the payloads at regular, user-configurable intervals to check the status of the payload. Payloads must respond and can also indicate availability of data or commands for the CCU.

Command Processing

Commands issued from remote applications are queued. When the CCU initiates any communication with the remote application to the remote application (for example, telemetry, polling data, mailbox check, ACKs), the CCU downloads and executes queued commands.

There are more than 50 Wave Glider commands. Wave Glider commands typically fall into one of the following functional categories:

- Make some change to the vehicle (for example, move rudder, turn a light on).
- Set system parameters (for example, telemetry rates).
- Request vehicle systems status (for example, battery level).
- Communicate with a payload.
- Define a course (for example, set/delete course Waypoints).
- Follow a course (for example, hold position, follow a set of Waypoints).

For more information see Chapter 5: *Commands*.

Command and Message Responses

When a command or payload message is received by the CCU, it will respond in one of two ways:

- Acknowledge (ACK) responses that let the requesting system know that the command or message was received and understood. The ACK response may also contain requested status information.
- Not-Acknowledge (NAK) responses that indicate the command or message was received but is in error, and could not be processed. The NAK response also indicates the general nature of the error.

If there is an additional command queued, it is downloaded to the vehicle, executed, and an ACK or NAK response is transmitted to remote applications. This process continues until there are no more commands in the queue.

If the communication channel is interrupted there may be multiple CCU data packets queued-up on the vehicle. When communication resumes, remote applications will see multiple incoming data packets. All communications are asynchronous, delayed, and in no consistent order. Any software designed for real-time processing of vehicle responses needs to consider this characteristic of communications with the CCU.

For more information see Chapter 6: *Vehicle Responses*.

CCU Communications Channels

The standard CCU communications channels are:

Liquid Robotics Proprietary and Confidential

- “Iridium Relay Link” (page 77)

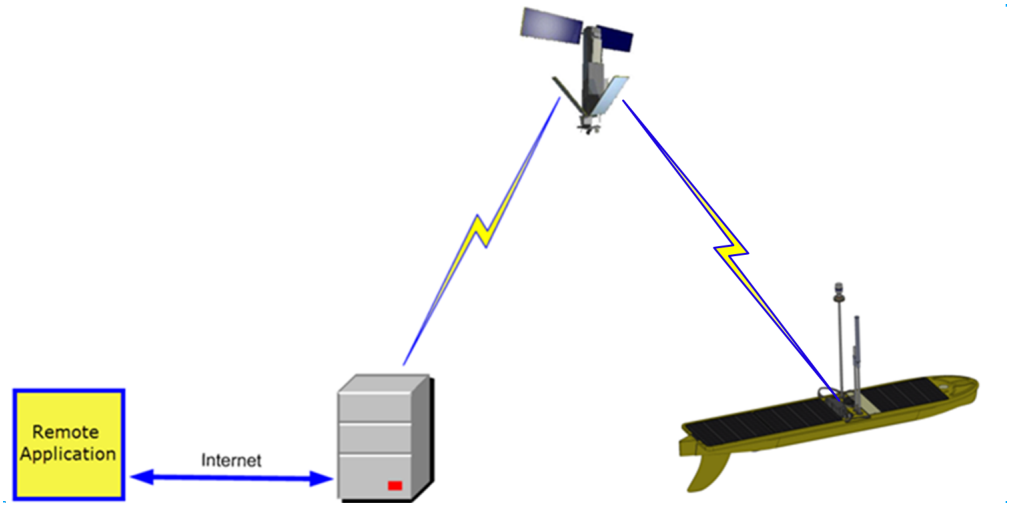
- Cellular communications - The Wave Glider SV3 CCU includes a cell modem that can be used for communications within cell range. The commands for turning the cell modem ON or OFF are available in the **WGMS SV3 Commands Window** from the **cellModem** device in the **Devices** menu.

- WiFi - The Wave Glider SV3 CCU includes a WiFi modem that can communicate directly with a WiFi enabled device. The commands for turning the WiFi modem ON or OFF are available in the **WGMS SV3 Commands Window** from the **wirelessAccessPoint** service in the **Services** menu.

Iridium Relay Link

The Iridium satellite network is an “always on” system with 100% global coverage. It is the primary communications link for the Wave Glider. The Wave Glider is typically linked to remote applications through an Internet connection to the Iridium satellite system. This is the standard means of controlling and monitoring the vehicle.

Figure 4.2. Vehicle to Remote Applications Iridium Communications



A Wave Glider’s Iridium modem sends and receives data over the Iridium satellite network using Iridium Short Burst Data (SBD) packet sessions. Iridium satellites exchange data with Iridium network servers in the Iridium data center. Iridium network servers use TCP/IP to exchange data with remote applications using the Liquid Robotics Iridium Relay.

Communication sessions are initiated by the Wave Glider CCU. The CCU periodically sends telemetry data packets to remote applications over the Iridium link. The time span between telemetry messages is user-configurable. Command data packets from remote applications are sent to the Iridium network servers, where they are queued until the Wave Glider initiates a communication session. After telemetry data is sent from the vehicle CCU to remote applications, any commands sent from remote applications are downloaded to the vehicle CCU.

When a command data packet has been processed by the CCU, it responds to all connected remote applications with an ACK (acknowledge) data packet containing the requested data. If the command requests some action by the vehicle (for example, moving the rudder), that action is also executed.

If the command data packet sent to the Wave Glider CCU is in some way incorrect, the vehicle responds with a NAK (not-acknowledged) data packet that indicates the nature of the error.

Liquid Robotics Proprietary and Confidential

Note

When a command is issued from a remote application over Iridium, there is a delay before the command is implemented on the Wave Glider. The duration of the delay depends on how frequently the Wave Glider CCU sends telemetry data, as well as the amount of telemetry data being transmitted. Iridium command data is queued until the Wave Glider CCU initiates a communication session. Iridium command data is not processed by the CCU until vehicle-initiated communications (telemetry intervals or ACK/NAK responses) are completed.

Short-Burst Data

The Iridium "Short-Burst-Data" (SBD) protocol is used for Iridium (Direct IP). SBD allows for economical transmission of relatively small data packets. Iridium refers to messages from the modem as "mobile-originated" (MO) and messages to the modem as "mobile-terminated" (MT).

The Iridium Short-Burst-Data format supports packets of limited size, depending on the modem used and the direction of the packet:

- Iridium 9601/2/3 modems can send messages (MO) of up to 340 bytes and receive messages (MT) of up to 270 bytes.
- Iridium 9522B and 9523 modems can send messages (MO) of up to 1960 bytes and receive message (MT) of up to 1890 bytes.

Iridium Message Buffering

Iridium service may occasionally be interrupted for up to several minutes. In this case, the CCU electronics will buffer any responses and telemetry, and then send out multiple responses and telemetry when service is resumed.

The message buffer is 16 responses deep. Once the 16-message limit is reached, any additional messages will be added to the buffer, and the oldest message discarded. Thus, for very long Iridium disruptions, some data may be lost.

CCU Communications Security

WGMS initiates the authentication process, requiring the Wave Glider to authenticate itself. The authentication process requires the exchange of the symmetric AES256 session key, then the Wave Glider and WGMS can exchange messages using a secure session.

CCU Communications Components

Figure 4.3 illustrates Iridium Relay, Cellular and WiFi communications configurations.

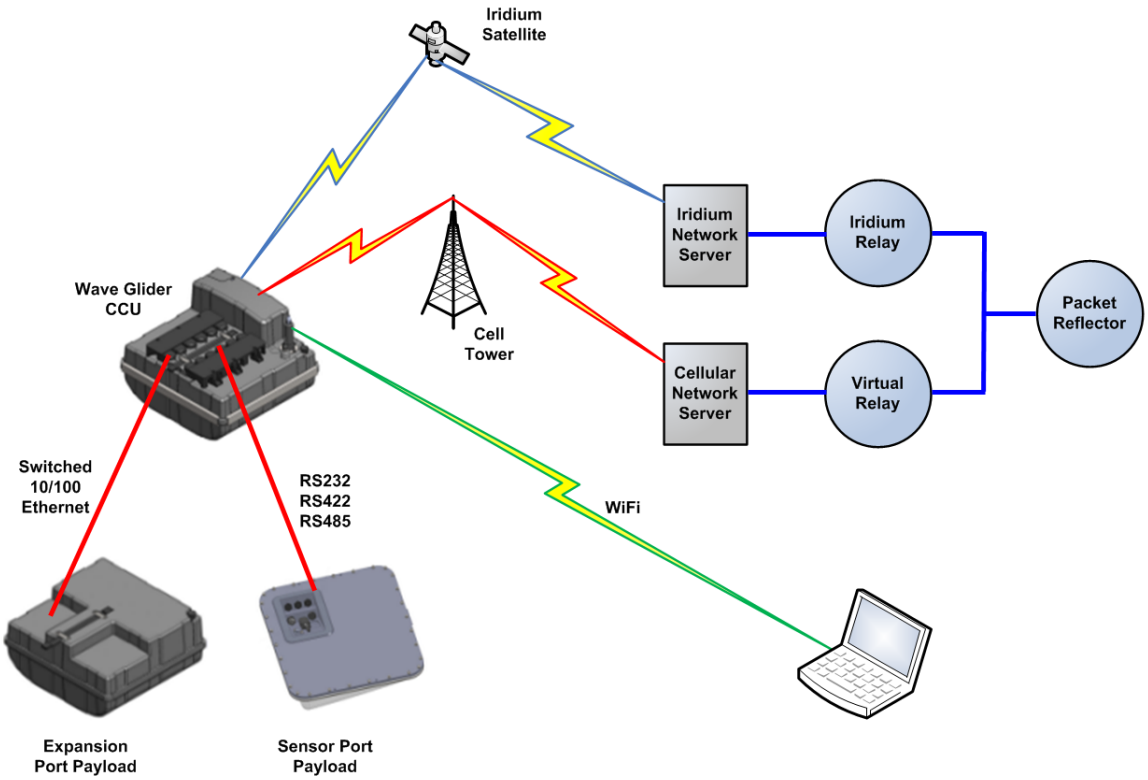
- Blue circles are user configurable Liquid Robotics-supplied components All of these components are co-located in a Liquid Robotics data center.
- Blue lines represent TCP/IP communications.

Specific deployments use a subset of the modules represented by the illustration, for example:

- Iridium Relay communications depend on an internet connection between the Iridium Relay and an Iridium network server. The connection between the Iridium network server and the CCU is over the Iridium satellite network.

- Cellular communications depend on an internet connection between the Virtual Relay and a cellular network server. The connection between the cellular network server and the CCU is over a cellular network.
- WiFi communications do not require an Internet connection. Data is transmitted directly between the CCU and a WiFi enabled device.

Figure 4.3. CCU Communications Components



The following sections describe the basic communications components of the Wave Glider system.

Communication Relays

Relays mask the variations between the underlying communications technologies by converting the communication interface they control to remote application compatible TCP/IP packets, or by converting TCP/IP packets from remote applications to a format that is compatible with a specific communication interface. Specific duties may include fragmenting and reassembling packets, managing queues and driving the physical interface.

Communication relays can be one of two types:

- Iridium Relay
- Virtual Relay

Iridium Relay

For most deployed Wave Gliders, the primary mode of communication is through the use of Short Burst Data (SBD) packets sent over the Iridium satellite constellation. Actual satellite communications with

Liquid Robotics Proprietary and Confidential

the vehicle are accomplished through a group of servers maintained by Iridium Communications, Inc. remote applications depends on the Iridium Relay to communicate with the Iridium Servers over the Internet, using TCP/IP. If necessary, data transfer can be done using Email, but at a lower data rate.

It is sometimes beneficial to communicate with the vehicle through means other than the Iridium Relay. Considerations can include Internet availability, bandwidth, cost, latency, privacy, security and distance. For example, in a payload development scenario inside a building, the Wave Glider's Iridium satellite antenna might not be able to see the sky, necessitating the use of WiFi.

Virtual Relay

A Liquid Robotics Virtual Relay Server is required when you want to establish a direct radio communications link between one or more Wave Gliders and one or more instances of a remote application. Examples of Wave Glider radio communications methods that require the Virtual Relay Server include: Cellular, Iridium RUDICS, and BGAN. Typically a Virtual Relay Server and a Packet Reflector are installed on a PC along with a remote application instance. A Virtual Relay Client is installed (on a SMC) on each Wave Glider vehicle. The Virtual Relay Server consists of a Virtual Relay Service that runs in the background, and a Virtual Relay Application that you use to configure and manage the Virtual Relay Service. When the Virtual Relay Service is started, it will continue running any time your PC is running, until you deliberately stop the service using the Stop Service function in the Virtual Relay Application. Regardless of the radio communications method, the TCP/IP protocol is used for data transmissions between a remote application and a vehicle. The Liquid Robotics Virtual Relay protocol rides on top of the TCP/IP protocol to support encryption and to ensure messages are received by their specified recipients. Communications between the Virtual Relay Server and a remote application are transmitted through a Packet Reflector in an Liquid Robotics Data Packet format over TCP/IP.

Standard Modems

The CCU on the Wave Glider vehicle includes an Iridium SBD modem to enable direct communications with the Iridium satellite network and a cell modem for communications with cellular networks. Similarly, for remote applications to communicate with Wave Glider vehicles in the absence of an Internet connection, the CCU includes a WiFi modem for direct communications between a Wave Glider and a WiFi enabled device.

Iridium Network Servers and Satellites

Iridium network servers are maintained in Iridium Communications, Inc. data centers. Iridium network servers relay data across the Iridium satellite network to complete communications between Wave Glider vehicles and remote applications. The Iridium satellite constellation can be accessed from the Internet over a TCP/IP link to an Iridium network server, or directly using an Iridium SBD modem.

CCU Communication Data Types

There are three CCU communication data types:

- Telemetry
- Commands
- Responses

Liquid Robotics Proprietary and Confidential

Telemetry

The vehicle automatically sends telemetry data packets at regular intervals. Telemetry Data indicates the position and the general status of the vehicle, as well as if there are any alarms. There are different telemetry data types, and each data type may be sent out on a different schedule. For more information on telemetry data types, see Chapter 7: *Vehicle Telemetry*.

Commands

When a remote application receives an Iridium packet (usually telemetry), it can then send a command to the vehicle. Likewise, if a payload on the vehicle receives a message, it can send a command to the CCU. The CCU then processes the command, if possible.

Commands generally fall into the following categories:

- Make some change to the vehicle (for example, move rudder, turn a light on).
- Define a course (for example, set course Waypoints).
- Follow a course (for example, hold position, follow a set of Waypoints).
- Set system parameters (for example, telemetry rates).
- Request vehicle systems status (for example, battery level).
- Route a message to a payload or to Iridium.

Responses

When a command or payload message is sent to the vehicle, the CCU will respond in one of two ways:

- Acknowledge (ACK) responses that let the requesting system know that the command was received. Depending on the type of command/message, the ACK response may also contain requested status information.
- Not-Acknowledge (NAK) responses that indicate that the data was received but is in error, and could not be processed. The NAK response also indicates the general nature of the error.

Communication Packets

All communication to and from the Wave Glider CCU is done with information packets containing a fairly small number of bytes (approx. 200 to 500 bytes). Note that all Relay data packets contain other packets that contain the message data and an error check.

Liquid Robotics Proprietary and Confidential

Figure 4.4. Level 3 Relay Data Packet Format

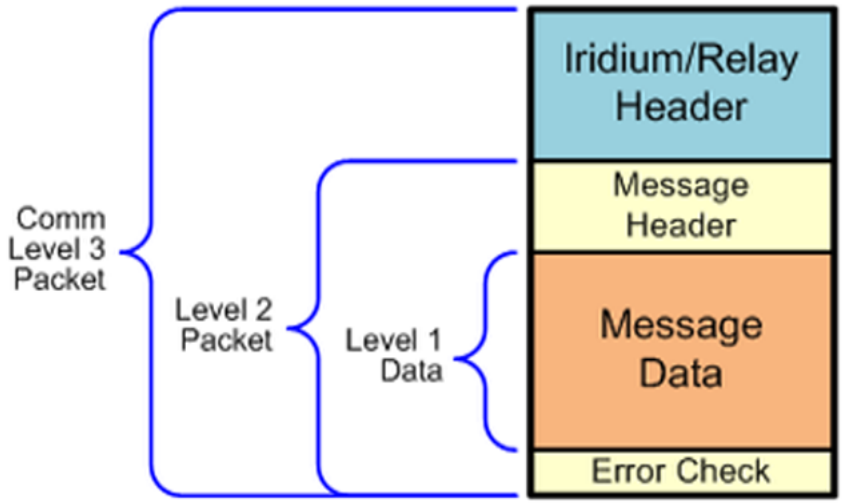
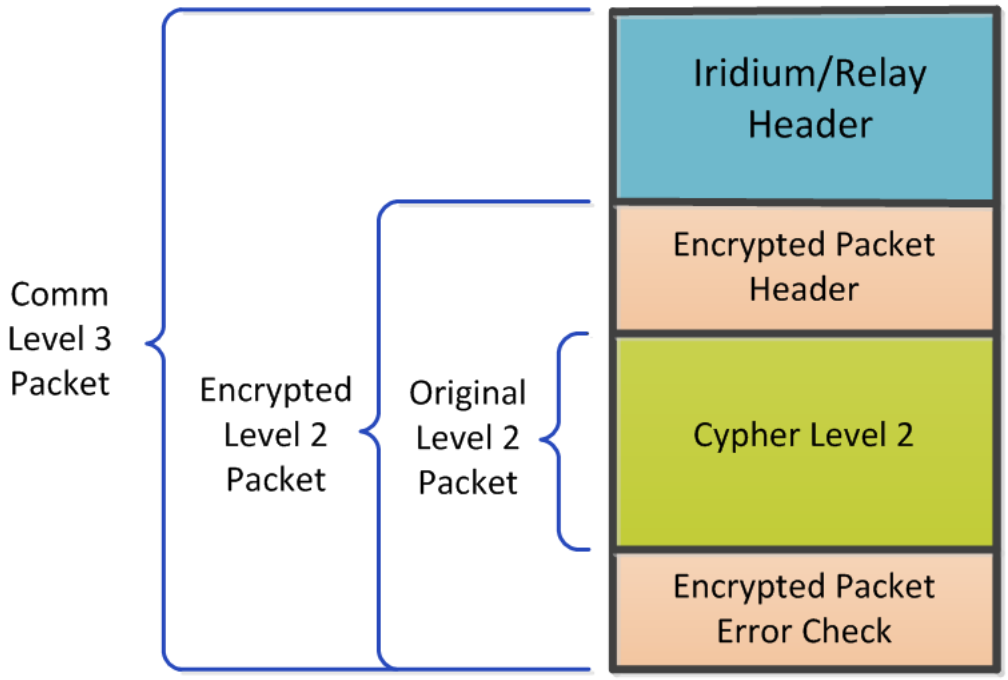


Figure 4.5. Level 3 Encrypted Relay Data Packet Format

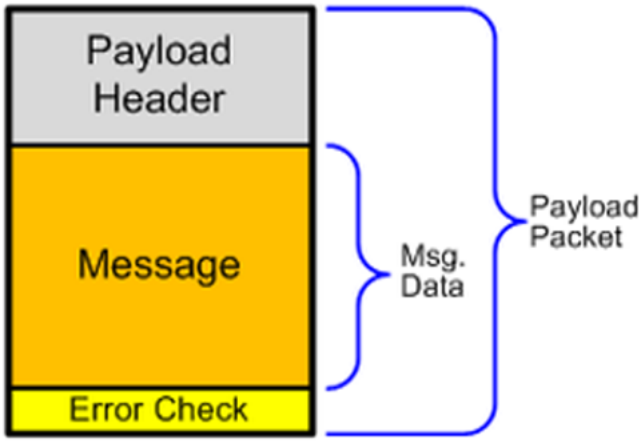


As shown in the figure, Relay data packets and encrypted Relay data packets both contain the same Level 2 data format.

Communication within or between payloads is done with payload data packets, but the message types and formats are different than what is used for Relay data packets.

Liquid Robotics Proprietary and Confidential

Figure 4.6. Level 2 Payload Data Packet Format



The Message in a payload data packet supports many different types of data, including data that can be put into a relay data packet. For more information on payload message data types, see Chapter 8: *CCU Payload Communications*.

All packet communications carry two CRC-16 bytes that can be checked by the receiving device. See Appendix F: *CRC-16 Calculation* for more information. All Iridium communications sent to remote applications have the error code checked on receipt. Payload message packets may or may not be checked, at the option of the message recipient.

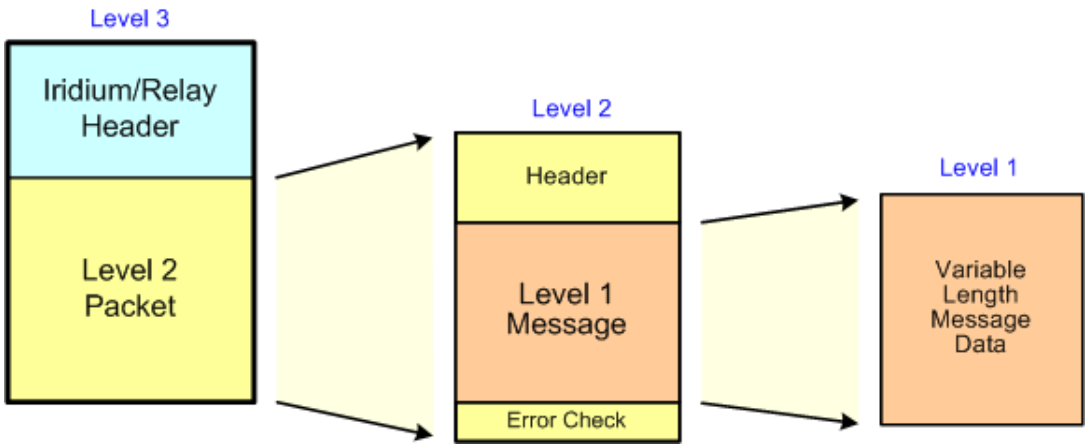
Relay Data Packets

Relay data packets are described as “Level 3”, “Level 2” and “Level 1”.

The Level 3 packet is used for communications to and from the vehicle. The Iridium Direct IP service inserts a header to report the ID number of the vehicle's modem (IMEI) along with other meta data relevant to the communication session. Relays also insert a header with identification numbers (Vehicle Unique ID) and meta data. These formats are described in Appendix E: *Relay Header Data*. In each case, Level 2 packet data can follow the Level 3 header.

The Level 2 packet contains its own header, message data, and an error check. The Level 1 packet can be raw data without any header. This hierarchy is shown in Figure 4.7.

Figure 4.7. Level 3, Level 2, and Level 1 Data Packets



Liquid Robotics Proprietary and Confidential

Level 2 packets within Relay data packets can be: Telemetry, Commands, or Responses and Alarms. The length and structure of the different packets will vary depending on the data type, and the intended recipient. These packets are described in more detail in Chapter 6: *Vehicle Responses*, Chapter 7: *Vehicle Telemetry*, and Chapter 8: *CCU Payload Communications*.

The details of the Relay data packet headers are managed by the CCU and by remote applications, so they need not be understood for payload development. For reference, Relay data packet header formats are described in Appendix E: *Relay Header Data*.

Addresses

Addresses are values used to route messages between the CCU, payloads and other devices (for example, sensors). Addresses are composed of a two-byte value. The MSB is the “Board ID” value which is usually put in Non-Volatile Memory on the board. The LSB is the “Task ID” which is usually assigned by the firmware on each board. For more information, see “Addresses” (page 153).

Vehicle Unique ID

The vehicle Unique ID is a unique 4-byte identification number stored in Non-Volatile Memory in the vehicle’s CCU. This number is assigned and uploaded when the vehicle is first brought online. It is used to identify the vehicle when data packets are sent by means other than Iridium.

Transaction ID

A Transaction ID is a 4 byte number assigned to each communication session (Command, ACK, NAK, and some types of Telemetry), between a remote application and a Wave Glider vehicle. The Transaction ID along with the Unique Vehicle ID and other meta data associated with the communication session provide a unique identifier for that transaction between a remote application and a specific vehicle. For more information see Chapter 6: *Vehicle Responses*, Chapter 7: *Vehicle Telemetry*, and Chapter 8: *CCU Payload Communications*.

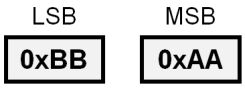
Communication Data Formats

Several of the message packets contain multi-byte representations of certain values. The formats of these message packets are defined below.

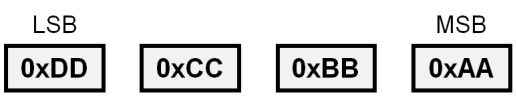
Little-Endian Byte Format

All multi-byte data values are represented in "little-endian" format. That is, values are stored with the least-significant byte first, and the most-significant byte last.

As an example of a value in 2-byte little-endian format, 0xAABB will be stored as:

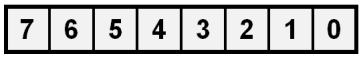


The 4-byte value 0xAABBCCDD will be stored as:



Liquid Robotics Proprietary and Confidential

Some data values are encoded in bit fields within bytes. The numbering of the bits is right-to-left, as shown in the figure below.



Waypoint Values

Each navigation waypoint is defined by a waypoint number, a Latitude value and a Longitude value.

The waypoint number is a single-byte value from 1 to 254 (the value zero cannot be used), and two waypoints cannot use the same number. However, the numbers are not required to be sequential, although there are advantages to defining a course using sequential numbers.

Latitude and Longitude use the WGS 84 coordinate system and must be converted to that coordinate system. Latitude and Longitude values are each stored as a 4-byte value (little-endian format). They are defined in units of 1/10,000 of an arc-minute, providing a resolution of about 0.2 meters. The sign (+ or –) indicates the direction as follows.

Latitude: (+) → North (–) → South [–54,000,000 to +54,000,000]

Longitude: (+) → East (–) → West [–108,000,000 to +108,000,000]

Heading Values

Heading values are in whole Degrees from 0 to 359. The value cannot be negative. Headings are stored as a 2-byte value in little-endian format.

Distance Values

Distances are indicated in meters, and are stored as 2-byte little-endian values.

Time Values

For any given command or response value, time values are specified in whole-number minutes, seconds or milliseconds.

- **Minutes:** Stored as a 2-byte value with a typical range of 1 to 1440 minutes. (1440 minutes = 24 hours).
- **Seconds:** Stored as a 1-byte value with a range of 1 to 255.
- **Milliseconds:** Stored as 1-byte value with a range of 1 to 255.

Heading Values

Heading values are in whole Degrees from 0 to 359. The value cannot be negative. Headings are stored as a 2-byte value in little-endian format.

Distance Values

Distances are indicated in meters, and are stored as a 2-byte value.

Liquid Robotics Proprietary and Confidential

Temperature Values

Temperature values are stored as a 2-byte signed value in units of °C / 10, thus giving 1/10th degree resolution.

File Names

A file name specified in a command field must be an 8+3, 11-byte ASCII string padded with spaces.

Only the characters 0x20 thru 0x7E are allowed (standard keyboard characters). The separating dot between the name and extension is omitted.

For example, DATA.TXT is stored as:

0	1	2	3	4	5	6	7	8	9	10
D	**A**	**T**	**A**					**T**	**X**	**T**

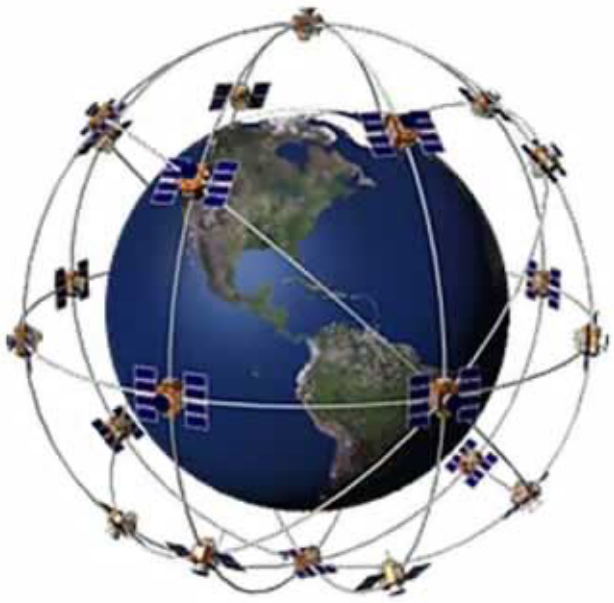
foo.c is stored as:

0	1	2	3	4	5	6	7	8	9	10
f	**o**	**o**						**c**		

GPS

The GPS constellation location signals are essential to vehicle operation. Most commands for Wave Glider positioning are based on Latitude and Longitude values. Telemetry and other status responses from the vehicle contain the Lat/Long location based on GPS data.

Figure 4.8. GPS Satellite Constellation



Liquid Robotics Proprietary and Confidential

GPS fixes are typically obtained every 1 seconds. However, there can be short periods (usually less than a minute) when no signal is received.

Navigation

Navigation of the vehicle is typically achieved by referencing Waypoints. Each Waypoint is defined by three values:

1. The Waypoint number - a user-assigned integer value from 1 to 254. No two Waypoints can have the same number.
2. A Latitude value - defined with a precision of arc-minutes / 10,000.
3. A Longitude value - defined with a precision of arc-minutes / 10,000.

The basic Navigation Modes of a vehicle are:

- Move to a Waypoint and maintain station there.
- Move from a start Waypoint through a sequence of Waypoints to an ending Waypoint.
- Continually move through a series of Waypoints, in a cyclic fashion.
- Maintain a constant heading.
- Hold a fixed rudder position.

For more information on waypoints and heading values, see “Communication Data Formats” (page 84).

Navigation Modes

Once a set of one or more waypoints have been defined, the vehicle is instructed to proceed in one of several navigation modes.

The navigation **Mode Names** and their associated **Values** are shown in the table below. The value is used in commands to specify the desired navigation mode. Vehicles return the navigation mode value in the telemetry as part of the command acknowledgement (ACK).

Table 4.1. Navigation Modes

Mode Name	Value
NAV_MODE_UNDEFINED	0x00
NAV_COURSE_LOOP	0x01
NAV_COURSE_HOLD	0x02
NAV_CUSTOM_LOOP	0x03
NAV_CUSTOM_HOLD	0x04
NAV_CUSTOM_HEADING	0x10
NAV_RUDDER_POSITION	0x20
NAV_TARGET_WAYPOINT	0x40
NAV_HOLD_POSITION	0x41

Liquid Robotics Proprietary and Confidential

Mode Name	Value
NAV_MODE_UNDEFINED_ALT_GPS	0x80
NAV_COURSE_LOOP_ALT_GPS	0x81
NAV_COURSE_HOLD_ALT_GPS	0x82
NAV_CUSTOM_LOOP_ALT_GPS	0x83
NAV_CUSTOM_HOLD_ALT_GPS	0x84
NAV_COURSE_LOOP_PID	0x0A
NAV_COURSE_HOLD_PID	0x0B
NAV_CUSTOM_LOOP_PID	0x0C
NAV_CUSTOM_HOLD_PID	0x0D
NAV_COURSE_LOOP_PID_ALT_GPS	0x8A
NAV_COURSE_HOLD_PID_ALT_GPS	0x8B
NAV_CUSTOM_LOOP_PID_ALT_GPS	0x8C
NAV_CUSTOM_HOLD_PID_ALT_GPS	0x8D
NAV_CUSTOM_HEADING_ALT_GPS	0x90
NAV_RUDDER_POSITION_ALT_GPS	0xA0
NAV_TARGET_WAYPOINT_ALT_GPS	0xC0
NAV_HOLD_POSITION_ALT_GPS	0xC1

Chapter 5. Commands

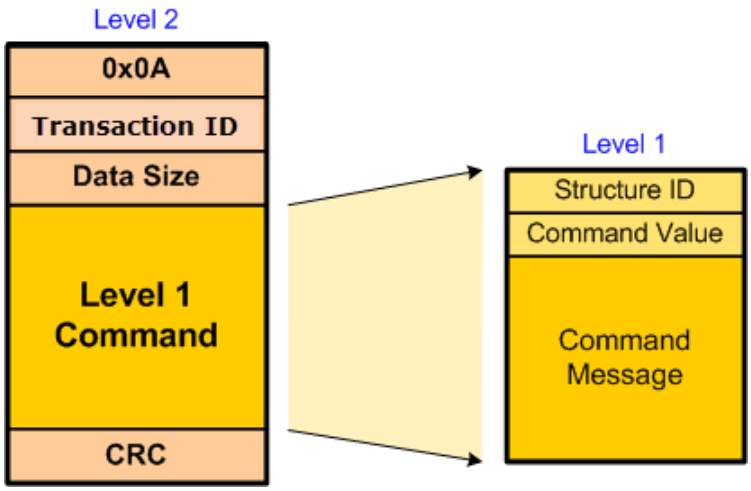
Commands are primarily used to specify some action by the vehicle, or to request information from the vehicle. Commands can be sent to the Wave Glider over the Iridium network, a cellular network, or directly using WiFi.

Command packets are received by the CCU and processed to remove any Level 3 transmission headers (Iridium or Relay). The CCU interprets the data in the binary command structure, and responds appropriately.

Command Data Packet Format

The Level 2 Command Packets have a leading header and a trailing CRC error check. In addition, the Command message itself always has a 3-byte header that defines the command type and structure.

Figure 5.1. Command Packet Format



Level 2 Command Packet	
0x0A	[1 byte] Indicates a Command Data Packet.
0xA7	[1 byte] Indicates an Encrypted Command Packet. Commands are encrypted by the relay and then sent to their destination over the communications channel. The header must be adjusted to reflect the changed size of the packet sent. Encrypted Command packets are transmitted between relays and vehicles in secure mode. A cipher level is implemented to transfer the LEVEL 2 data packet. Type is maintained to identify the direction of the packet (from/to vehicle) and counts are required to distinguish between the level 2 cipher data and the AES initialization vector data in the packet. If IV is not included (size = 0) then IV is assumed to be the previously used IV. Figure 4.5 illustrates the level 3 encrypted relay data packet format.
Transaction ID	[4 bytes] A Liquid Robotics-defined ID stored in a remote application database. For more information see “Transaction ID” (page 84).

Liquid Robotics Proprietary and Confidential

Level 2 Command Packet	
Size	[2 bytes] The size of the Command message that follows.
Command	[max 217 bytes] The Level 1 Command Packet. This packet always starts with a Structure ID and a Command Value.
CRC	[2 bytes] The CRC16 of all bytes in the data packet (except the two CRC bytes).

Level 1 Command Message	
Structure ID	[1 byte] Defines the expected size and format of the Command Data.
Command Value	[2 bytes] A number specific to the Command Data type. Tells the CCU computer the type of command.
Command Message	Variable-length field containing the actual command.

Command values and structure IDs are summarized in the table below. For command data descriptions, see Appendix A: *Command Descriptions*.

Command Summary

There are two categories of commands for Wave Glider SV3:

- Commands that can be executed on SV3 Vehicles only.
- SV2 commands that can be executed on SV3 Vehicles.

Commands for SV3 Vehicles Only

The SV3 commands that are supported on each SV3 Vehicle are included in the Commands Manifest for that Vehicle. Typically the SV3 Commands Manifest is parsed to populate a GUI, for example, the SV3 Commands Window in WGMS. For more information see "SV3 Manifests" (page 228) and "Commands for SV3 Vehicles Only" (page 265).

SV2 Commands for SV3 Vehicles

Table 5.1 lists the SV2 commands that run on SV3 Vehicles. Each command listed in the table includes a link to a detailed description that is provided in Appendix A: *Command Descriptions*. If an unsupported SV2 command is issued to an SV3 Vehicle, an error message will be returned.

Table 5.1. Command Summary

Command Name	Command Value	Structure ID	Link to Command Description Appendix
Hold Current Position	**0x0001**	**0x10**	See "Hold Current Position (0x0001)" (page 268)
Hold Station At Waypoint	**0x0002**	**0x10**	See "Hold Station at Waypoint (0x0002)" (page 268)
Follow Sequential Waypoint Course	**0x0003**	**0x10**	See "Follow Sequential Waypoint Course (0x0003)" (page 269)
Right Rudder	**0x0004**	**0x10**	See "Rudder Right/Left/Center (0x0004, 0x0005, 0x0006)" (page 269)

Liquid Robotics Proprietary and Confidential

Command Name	Command Value	Structure ID	Link to Command Description Appendix
Left Rudder	**0x0005**	**0x10**	See "Rudder Right/Left/Center (0x0004, 0x0005, 0x0006)" (page 269)
Center Rudder	**0x0006**	**0x10**	See "Rudder Right/Left/Center (0x0004, 0x0005, 0x0006)" (page 269)
Follow Fixed Heading	**0x0008**	**0x10**	See "Follow Fixed Heading (0x0008)" (page 270)
Set Rudder Position	**0x1006**	**0x10**	See "Set Rudder Position (0x1006)" (page 270)
Visible Light Off/On	**0x2001 0x2002**	**0x10**	See "Visible Light (Marker Light) Off/On (0x2001, 0x2002)" (page 270)
Retrieve Status	**0x4001**	**0x10**	See "Retrieve Status (0x4001)" (page 271)
Set Status	**0x4002**	**0x11**	See "Set Status (0x4002)" (page 271)
Retrieve Identification	**0x4003**	**0x10**	See "Retrieve Identification (0x4003)" (page 272)
Retrieve Power Status	**0x5001**	**0x10**	See "Retrieve Power Status (0x5001)" (page 273)
New Waypoint	**0x8000**	**0x10**	See "New Waypoint (0x8000)" (page 273)
Read Waypoint	**0x8003**	**0x10**	See "Read Waypoint (0x8003)" (page 273)
Follow Course and Loop	**0x8101**	**0x12**	See "Follow Course and Loop (0x8101)" (page 274)
Follow Course With Options	**0x8102**	**0x12**	See "Follow Course With Options (0x8102)" (page 275)
Read File Request	**0x9003**	**0x1B**	See "Read File Request (0x9003)" (page 275)
Read File Transaction Closed	**0x9007**	**0x10**	See "Read File Transaction Closed (0x9007)" (page 276)
Write Console Script	**0x9100**	**0x25**	See "Write Console Script (0x9100)" (page 277)
Float Reboot	**0xA000**	**0x31**	See "Float Reboot (0xA000)" (page 277)

Chapter 6. Vehicle Responses

Most response messages are ACK/NAK data packets sent in response to Commands.

ACK/NAK Protocol

Commands sent to the CCU will be acknowledged with an ACK or NAK. The ACK/NAK response contains the command ID to allow the receiving system to track the command being acknowledged, even if a data transmission from the vehicle is lost.

Responses are as follows:

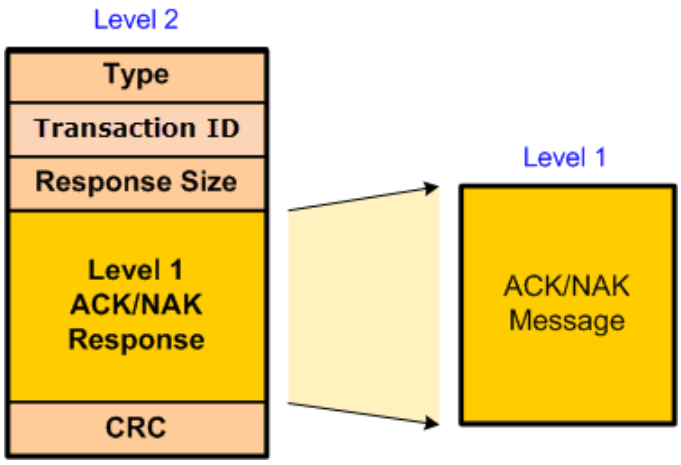
- ACK signals successful packet reception and acceptance of its content.
- NAK signals successful packet reception but rejection of its content.
- No response (timeout) signals non-reception of a packet.

Note that the "Float Reboot" command is not expected to ACK or NAK.

ACK/NAK Packet Format

The ACK/NAK response packet format is shown in Figure 6.1. Note that, unlike command packets, the Level 1 ACK/NAK message packet has no standard header structure. The different ACK/NAK "Type" values indicate the format of the Level 1 packet.

Figure 6.1. ACK/NAK Response Packet Format



Where:

Type: [1 byte] ACK or NAK Data Type. For more information see "ACK Data Type Numbers" (page 93) and "NAK Data Type Numbers" (page 93).

Transaction ID: [4 bytes] The Transaction ID value for the communication session. For more information, see "Transaction ID" (page 84).

Size: [2 bytes] Size of the ACK or NAK Message that follows. Size = 0 → No Data.

ACK/NAK Response: [max 196 bytes] Level 1 ACK or NAK data/message return, if any.

Liquid Robotics Proprietary and Confidential

CRC: [2 bytes] CRC16 of all previous bytes in the Level 2 ACK/NAK packet.

ACK Data Type Numbers

Allowed Data Types for the ACK return are:

Type	ACK Return Payload	Reference
0x10	Status With Waypoint Read-Back	See "ACK - Status with Waypoint Read-Back (0X10)" (page 94)
0x11	Status Return	See "ACK - Status Return (0x11)" (page 95)
0x12	Identification Data	See "ACK - Vehicle Identification Data (0x12)" (page 96)
0x13	No Payload	See "ACK - No Payload Response (0x13)" (page 99)
0x14	Power Controller Status	See "ACK - Power Controller Status (0x14)" (page 99)
0x1B	Read File	See "ACK - Read File Block (0x1B)" (page 100)
0x3D	Console Data	See "ACK – Console Data (0x3D)" (page 101)
0xA1	Reserved	

NAK Data Type Numbers

Allowed data types for the NAK return are:

Type	NAK Type	Reference
0x20	(reserved)	
0x21	(reserved)	
0x22	Invalid Command	See "NAK - Invalid Command (0x22)" (page 101)
0x23	Invalid Data	See "NAK – Invalid Data (0x22)" (page 101)
0x24	File Error	See "NAK – File Error (0x24)" (page 101)
0x25	Security Failure	See "NAK – Security Failure (0x25)" (page 102)
0x26	Console Data	See "NAK – Console Data (0x26)" (page 102)
0x27	NAK with Description	See "NAK – With Description (0x27)" (page 103)

Unsolicited Response Type Numbers

Allowed data Types for an unsolicited response are:

Liquid Robotics Proprietary and Confidential

Type	Message Type	Reference
0x60	New File Available	See "New File Available (0x60)" (page 103)
0x1D	Write Console Output to Client	See "Send Console Output to Client (0x1D)" (page 104)

Relay Messages

These are messages from a Relay to a remote application.

Type	Message Type	Reference
0x70	Relay Status	See "Relay Status (0x70)" (page 104)

ACK Response Data

The formats of data returned in the ACK packets are described below.

ACK - Status with Waypoint Read-Back (0X10)

This type of Status packet is sent in response to the commands:

New Waypoint - see "New Waypoint (0x8000)" (page 273)

Read Waypoint - see "Read Waypoint (0x8003)" (page 273)

The format is identical to the regular "Status Return" - see "ACK - Status Return (0x11)" (page 95), except for the last two fields (Waypoint Lat/Long).

Field Name	Bytes Used	Byte / Bit	Value [range] (units)	Default	Note
Follow Course	1	0 / 0	0 → not, 1 → Enabled	**1**	[1]
Manual Navigation		0 / 1	0 → not, 1 → Enabled	**0**	[1]
Follow Heading		0 / 2	0 → not, 1 → Enabled	**0**	[1]
Hold Station		0 / 3	0 → not, 1 → Enabled	**0**	[1]
(unused)		0 / 4	0		
XBee State **(SV2 Only)**		0 / 5	0 → Off, 1 → On		
Marker Light State		0 / 6	0 → Off, 1 → On		
IR Light State		0 / 7	0 → Off, 1 → On		
Telemetry Rate	2	1-2	[1 – 1440] (minutes)	**2**	
Low Battery Telemetry Rate	2	3-4	[1 – 1440] (minutes)	**20**	
Alarm Telemetry Rate	2	5-6	[1 – 30] (minutes)	**2**	
(reserved)	1	7	0		
Sub-to-Float Timeout	1	8	[20 – 200] (millisecond)	**20**	

Liquid Robotics Proprietary and Confidential

Field Name	Bytes Used	Byte / Bit	Value [range] (units)	Default	Note
(reserved)	2	9-10	0		
Float Low Battery	1	11	[1 – 65] (10 Wh)	**20**	
Float Critical Battery	1	12	[1 – 65] (10 Wh)	**3**	
(reserved)	3	13-15	0		
Float Pressure Threshold High	1	16	[0 – 255] (kPa)	**255**	
Float Pressure Threshold Low	1	17	[0 – 255] (kPa)	**115**	
Float Temperature Threshold	1	18	[0 – 255] (°C)	**70**	
Waypoint Arrival Distance	2	19-20	[5 – 5000] (meters)	**20**	
(reserved)	1	21	0		
Power Controller Telemetry Rate	2	22-23	[5 – 2880] (minutes)	**60**	
(reserved)	2	24-25	0		
Float-to-Sub Comm Period	1	26	[1 – 240] (seconds)	**15**	
Waypoint Latitude	4	27-30	[See “Waypoint Values” (page 85)]		
Waypoint Longitude	4	31-34	[See “Waypoint Values” (page 85)]		

ACK - Status Return (0x11)

This type of Status packet is sent in response to the commands:

Set Status - see “Set Status (0x4002)” (page 271)

Retrieve Status - see “Retrieve Status (0x4001)” (page 271)

Hold Current Position- see “Hold Current Position (0x0001)” (page 268)

Note that this packet contains read-only fields that are not used in the **Set Status** command.

Field Name	Bytes Used	Byte/Bit	Value Range (units)	Default	Notes
Follow Course [1]	1	0 / 0	0 → not, 1 → Enabled	**1**	[1]
Manual Navigation [1]		0 / 1	0 → not, 1 → Enabled	**0**	[1]
Follow Heading [1]		0 / 2	0 → not, 1 → Enabled	**0**	[1]
Hold Station [1]		0 / 3	0 → not, 1 → Enabled	**0**	[1]
(unused)		0 / 4	0		
XBee State **(SV2 Only)**		0 / 5	0 → Off, 1 → On		
Marker Light State		0 / 6	0 → Off, 1 → On		

Liquid Robotics Proprietary and Confidential

Field Name	Bytes Used	Byte/Bit	Value Range (units)	Default	Notes
IR Light State		0 / 7	0 → Off, 1 → On		
Telemetry Rate	2	1	[1 – 1440] (minutes)	**2**	
Low Battery Telemetry Rate	2	3	[1 – 1440] (minutes)	**20**	
Alarm Telemetry Rate	2	5	[1 – 30] (minutes)	**2**	
(reserved)	1	7	0		
Sub Float Timeout	1	8	[20 – 200] (millisecond)	**20**	
(reserved)	2	9	0		
Float Low_Battery Power Level	1	11	[1 – 65] (10 Wh)	**20**	
Float Critical_Battery Power Level	1	12	[1 – 65] (10 Wh)	**3**	
(reserved)	3	13	0		
Float Pressure_Threshold High	1	16	[0 – 255] (kPa)	**255**	
Float Pressure_Threshold Low	1	17	[0 – 255] (kPa)	**115**	
Float Temperature Threshold	1	18	[0 – 255] (°C)	**70**	
Waypoint Arrival Distance	2	19	[5 – 5000] (meters)	**20**	
(reserved)	1	21	0		
Power Controller Telemetry Rate	2	22	[5 – 2880] (minutes)	**60**	
(reserved)	2	24	0		
Float-to-Sub Comm Period	1	26	[1 – 240] (seconds)	**15**	
(Hold Position Latitude)	4	27	0 [See “Waypoint Values” (page 85)]		[2]
(Hold Position Longitude)	4	31	0 [See “Waypoint Values” (page 85)]		[2]

NOTES:

[1] Only one of these bits will be enabled at a time.

[2] Latitude and Longitude values are valid only when ACKing the "Hold Current Position" command. For the “Set Status” and “Retrieve Status” commands, the response returns a value of zero.

ACK - Vehicle Identification Data (0x12)

This response contains data on the vehicle software and hardware. It is sent in response to the command:

Retrieve Identification - see “Retrieve Identification (0x4003)” (page 272)

Liquid Robotics Proprietary and Confidential

All Version numbers are stored as [Major] [Minor], using 2 bytes (1 byte each).

Identification Packet Field Name	Bytes Used	Byte #	Value [range] (units)
Hour	1	0	[0 – 23] (hours)
Minute	1	1	[0 – 59] (minutes)
Second	1	2	[0 – 59] (seconds)
Month	1	3	[1 – 12]
Day	1	4	[1 – 31]
Year	1	5	[0 – 99] (20xx)
Float_C&C Version	2	6-7	[major] [minor]
Float_C&C Build Number	2	8-9	[1 – 255]
Reported IMEI	15	10-24	[15 digits]
Sub Firmware Version	2	25-26	[major] [minor]
Sub Hardware Version	2	27-28	[major] [minor]
Sub Firmware CRC-32	4	29-32	
Sub EEPROM CRC-32	4	33-36	
Sensor Firmware Version	2	37-38	[major] [minor]
Sensor Hardware Version	2	39-40	[major] [minor]
Sensor Firmware CRC-32	4	41-44	
Sensor EEPROM CRC-32	4	45-48	
Bubble Firmware Version **(SV2 Only)**	2	49-50	[major] [minor]
Bubble Hardware Version **(SV2 Only)**	2	51-52	[major] [minor]
Bubble Firmware CRC-32 **(SV2 Only)**	4	53-56	
Bubble EEPROM CRC-32 **(SV2 Only)**	4	57-60	
Batteries Firmware Version	2	61-62	[major] [minor]
Batteries Hardware Version	2	63-64	[major] [minor]
Batteries Firmware CRC-32	4	65-68	
Batteries EEPROM CRC-32	4	69-72	
Untwist Firmware Version	2	73-74	[major] [minor]

Liquid Robotics Proprietary and Confidential

Identification Packet Field Name	Bytes Used	Byte #	Value [range] (units)
Untwist Hardware Version	2	75-76	[major] [minor]
Untwist Firmware CRC-32	4	77-80	
Untwist EEPROM CRC-32	4	81-84	
COM Spec Version	2	85-86	[major] [minor]
COM Spec Revision	1	87	
Iridium Modem Revision	2	88-89	
XBee Channel **(SV2 Only)**	1	90	[12 to 23]
XBee PAN ID **(SV2 Only)**	2	91-92	
XBee Address **(SV2 Only)**	2	93-94	
(reserved)	4	95-101	
Count of Program Board Records	2	102-103	**N**
Record #1	17	104-120	[See "Program Board Records" below]
Record #2	17	121-137	[See "Program Board Records" below]
. . . . .	. . .	. . .	
. . . . .	. . .	. . .	
. . . . .	. . .	. . .	
Record #N	17		[See "Program Board Records" below]

Program Board Records have the following structure:

Field Name	Bytes	Value [range]	Note
Storage Controller Address	2		[1]
Board Address	2		[2]
Selected Program Transaction ID	4		[3]
Default Program Transaction ID	4		[4]
Currently Reported Program Transaction ID	4		[5]
Has Files	1	0x00 or 0xFF	[6]

Notes:

[1] The "Storage Controller Address" is the location where the program originates.

[2] The "Board Address" is the board where the program executes.

[3] The "Selected Program" is the designated standard program to run.

[4] The "Default Program" is the fall-back program.

[5] The "Currently Reported Program" is the one that last ran.

[6] Boolean value indicates whether any files are stored on the board.

0x00 → No

0xFF → Yes (board is a Storage Controller)

ACK - No Payload Response (0x13)

This ACK contains no payload data (Size = 0) and is sent in response to vehicle action commands:

Follow Sequential Waypoint Course - see "Follow Sequential Waypoint Course (0x0003)" (page 269)

Follow Fixed Heading - see "Follow Fixed Heading (0x0008)" (page 270)

Follow Course and Loop - see "Follow Course and Loop (0x8101)" (page 274)

Follow Course With Options - see "Follow Course With Options (0x8102)" (page 275)

Rudder Left/Right/Center - see "Rudder Right/Left/Center (0x0004, 0x0005, 0x0006)" (page 269)

Light, IR, Off/On - see "Visible Light (Marker Light) Off/On (0x2001, 0x2002)" (page 270).

Read File Transaction Closed - see "Read File Transaction Closed (0x9007)" (page 276)

ACK - Power Controller Status (0x14)

Sent in response to the command: **Retrieve Power Status** - see "Retrieve Power Status (0x5001)" (page 273)

Field Name	Bytes Used	Byte #	Value [range] (units)
Hour	1	0	[0 – 23] (hours)
Minute	1	1	[0 – 59] (minutes)
Second	1	2	[0 – 59] (seconds)
Month	1	3	[1 – 12]
Day	1	4	[1 – 31]
Year	1	5	[20xx]
Payload_1 Current	1	6	(10 mA)
Payload_2 Current	1	7	(10 mA)
Umbilical Current	1	8	(10 mA)
Untwist Current	1	9	(10 mA)
Total Power Remaining	2	10-11	(10 mWh)

Liquid Robotics Proprietary and Confidential

Field Name	Bytes Used	Byte #	Value [range] (units)
Battery Current	2	12-13	[signed] (10 mA)
System Current	2	14-15	(10 mA)
System Voltage	2	16-17	(1 mV)
Solar_Power_1 Indicator	1	18	[0 – 255] 0 → Off
Solar_Power_2 Indicator	1	19	[0 – 255] 0 → Off
Solar_Power_3 Indicator	1	20	[0 – 255] 0 → Off
Solar_Power_4 Indicator	1	21	[0 – 255] 0 → Off
Battery_0 Status	8	22-29	[see "Battery Status" below]
Battery_1 Status	8	30-37	[see "Battery Status" below]
Battery_2 Status	8	38-45	[see "Battery Status" below]
Battery_3 Status	8	46-53	[see "Battery Status" below]
Battery_4 Status	8	54-61	[see "Battery Status" below]
Battery_5 Status	8	62-69	[see "Battery Status" below]
Battery_6 Status	8	70-77	[see "Battery Status" below]

Battery Status	Bytes	Bit	Value [range] (units)
Battery Present	1	0	0 → No 1 → Yes
Battery Discharged		1	0 → No 1 → Yes
Battery Charged		2	0 → No 1 → Yes
Battery Discharging		3	0 → No 1 → Yes
Battery Terminate_Discharge		4	0 → No 1 → Yes
Battery Terminate_Charge		5	0 → No 1 → Yes
Battery Over_Temp		6	0 → No 1 → Yes
Battery Online		7	0 → No 1 → Yes
Battery Temp	1		[-10 to 100] (°C)
Battery Power_Remaining	2		(10 mWh)
Battery Current	2		(10 mA)
Battery Voltage	2		(1 mV)

ACK - Read File Block (0x1B)

Sent in response to the command:

Read File Request - see “Read File Request (0x9003)” (page 275)

If the file length exceeds the length of the data field, there may be multiple responses to a single “Read File Request” command.

Field Name	Bytes	Value
Storage Controller Address	2	Matches command request
Offset (Seek)	4	Value specified in request

Liquid Robotics Proprietary and Confidential

Field Name	Bytes	Value
EOF Flag	1	0x00 = More responses will be sent 0xFF = End of file, no more responses
Number of Bytes in Data	2	**N** [≥ 1]
Data	1 to N	File data block

ACK – Console Data (0x3D)

The **Write Console** commands can have ACKs that carry some text in the response.

Field Name	Bytes	Value
Format	1	0=ASCII format and 1=binary format
Disposition	1	3 = Succeeded TBD
Number of Bytes in Data	2	The size of the Data field in bytes
Data	1..N	ASCII data to be written into Console or ASCII/ binary data results from a command invocation

NAK Response Data

NAK - Invalid Command (0x22)

The command code number was not recognized.

This NAK contains no payload data (Size = 0).

NAK – Invalid Data (0x22)

The data in the command is not correct.

This NAK contains no payload data (Size = 0).

NAK – File Error (0x24)

Response given for a file error in the commands:

Read File Request - see "Read File Request (0x9003)" (page 275)

The NAK message returns one or more disposition codes indicating the type of error.

Field Name	Bytes	Value [range]	Notes
Count of Dispositions	2	**N** [≥ 1]	
Disposition 1	1	[3 to 17]	[1]
. . . . .	. . .	. . . . .	
. . . . .	. . .	. . . . .	

Liquid Robotics Proprietary and Confidential

Field Name	Bytes	Value [range]	Notes
Disposition N	1	[3 to 17]	

Note:

[1] Disposition values are:

3 = Some items failed, others succeeded

5 = Bad data

7 = File exists but is marked for no-change

8 = File would be overwritten

9 = File is read-only

10 = Missing storage device

11 = Failed CRC-16 check. File was deleted if requested.

12 = Failed MD5 check. File was deleted if requested.

13 = Storage full

14 = Storage I/O error

15 = File control structures corrupted

16 = Program selection configuration on vehicle is corrupt

17 = The number of active file commands exceeds handling capability

NAK – Security Failure (0x25)

When encryption negotiations fail the vehicle will return this NAK.

NAK – Console Data (0x26)

The **Write Console** commands can have NAKs that contain the reason why the console command failed in the response.

Field Name	Bytes	Default Value
Count of Dispositions	2	1
Disposition	1	1 = Command OK 2 = Response Truncated 10 = Unknown Command 11 = Device Busy 12 = Unsupported Command 13 = Empty Command

Liquid Robotics Proprietary and Confidential

		14 = Wrong Device 15 = Command Failed 16 = Device Not Responding 20 = Command Not Supported 30 = Invalid Command Length 31 = Unknown Destination

NAK – With Description (0x27)

NAK that includes an explanation of why a command was not executed.

Field Name	Bytes	Default Value
Length in Bytes	2	n
Sequence Number	1	Sequence number for multi-nack description
N bytes of ASCII string	n	Description returned from the vehicle

Unsolicited Messages

These Messages can be sent by the vehicle without being prompted by a Command.

New File Available (0x60)

The content has following format:

Field Name	Bytes	Value	Note
Storage Controller Address	2		[1]
File Name	11	8+3 ASCII string - see “File Names” (page 86)	
File Size	4	Bytes	
Creation Date	6	Date + Time	[2]
Modified Date	6	Date + Time	[2]
File Attributes	1	0x01 bit = 1 → read-only 0x80 bit = 1 → MD5 check was successful All other bits are user-defined	
Provided CRC-16	2		
Provided MD5	16		
Calculated CRC-16	2		[3]
Calculated MD5	16		[3]

Notes:

[1] The "Storage Controller Address" is the 1-byte Board Address, and the 1-byte Task ID - see “Addresses” (page 84). The Float CCU has the reserved address value: **0x0000**

[2] Dates use 1 byte per value in the order shown below. Value ranges are shown in [ ].

Hour	[0 to 23]
Minute	[0 to 59]
Second	[0 to 59]
Month	[1 to 12]
Day	[1 to 31]
Year-2000	[0 to 99]

[3] The "Calculated" CRC and MD5 are the values calculated on a previous file access command. If they have not been calculated, the default value is 0.

Send Console Output to Client (0x1D)

Sends a stream of ASCII data from the vehicle to remote applications.

Field Name	Bytes	Value [range]	Notes
Source Address	2	[1 to 256] + [1 to 256]	[1]
Function	1	Program-defined	[2]
Number of Bytes in Data	2	**N** [≥ 1]	
Data	1 to N	ASCII data being sent to the Console	

Notes:

[1] The "Source Address" is the address of the board sending the data. Float CCU = 0x0000.

[2] The "Function" is a value defined by the vehicle program. It can be used by the Client.

Relay-Only Message

Certain commands can only be issued by a relay to remote applications. These typically are instructions to remote applications that are sent when the vehicle connects to the relay.

Relay Status (0x70)

This is sent by a Relay to remote applications. It provides the characteristics and the operating status of the Relay.

Field Name	Bytes	Byte #	Bit #	Value (default)	Notes
Channel State	1	0	0	0 or 1	[1]
Message Not Sent Alarm			1	0 or 1	[2]

Liquid Robotics Proprietary and Confidential

Field Name	Bytes	Byte #	Bit #	Value (default)	Notes
Communications Error Alarm			2	0 or 1	[3]
(reserved)			3	0	
(reserved)			4	0	
(reserved)			5	0	
(reserved)			6	0	
(reserved)			7	0	
Channel Revision Major	1	1			
Channel Revision Minor	1	2			
Channel Revision Sub-Minor	1	3			
Date	6	4 - 9			[4]

Notes:

[1] The "Channel State" values indicate the following: 0 → Channel not active 1 → Channel active

[2] The "Message Not Sent Alarm" indicates: Relay tried to forward message to the vehicle, but the file was not accepted.

[3] The "Communications Error Alarm" indicates: The Relay was getting errors while trying to communicate with the vehicle.

[4] Dates use 1 byte per value in the order shown below. Value ranges are shown in [ ].

Hour	[0 to 23]
Minute	[0 to 59]
Second	[0 to 59]
Month	[1 to 12]
Day	[1 to 31]
Year-2000	[0 to 99]

Vehicle File System Responses

New File Available

This is a message from the vehicle to the remote application letting the operator know that there is a new file available.

Field Name	Bytes	Size
Storage Controller Address	2	Any (0=Float CCU)
File Name	11	ASCII string 8.3 padded file name
File Size	4	

Liquid Robotics Proprietary and Confidential

Field Name	Bytes	Size
Creation Date	6	Hour (1 byte 0..23) Minute (1 byte, 0..59) Second (1 byte, 0..59) Month (1 byte, , 1..12) Day (1, 1..31) Years since 2000 (1 byte, 0..255)
Modified Date	6	Hour (1 byte 0..23) Minute (1 byte, 0..59) Second (1 byte, 0..59) Month (1 byte, , 1..12) Day (1, 1..31) Years since 2000 (1 byte, 0..255)
File Attributes	1	0
Provided CRC-16	2	Any
Provided MD-5	16	Any
Calculated CRC-16	2	Any
Calculated MD-5	16	Any

Read File Block ACK

This is how the vehicle transfers files to the remote application after a **Read File Request** command is successfully issued.

Field Name	Bytes	Default Value
Storage Controller Address	2	Any (0=Float CCU)
Offset (Seek)	4	Taken from request
EOF	1	0 = more responses will be sent 0xFF = end of file, no more responses to be sent
Number of Bytes in Data	2	The length of Data field (> 0)
Data	0..N	Data accompanying this message

Chapter 7. Vehicle Telemetry

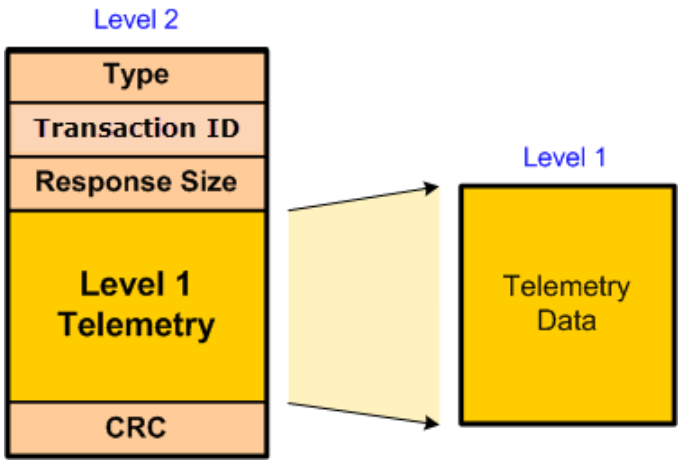
Telemetry data packets from standard Vehicle sensors (excluding the Resilient Tracker), and from the Vehicle power system, are automatically sent by the CCU at the rates defined by the Set Status command. For more information see "Set Status (0x4002)" (page 271). Note that the "Standard" Telemetry rate can differ from the rate for the Retrieve Power Status command. For more information see "Retrieve Power Status (0x5001)" (page 273). For information on Resilient Tracker telemetry, see the *Wave Glider SV3 Integrated Sensor User Guide for the Xeos Technologies Rover (Resilient Tracker)*.

Telemetry can be sent to an off-vehicle location using an Iridium, Cellular or WiFi communications channel, or it can be sent to a payload. For more information, see Chapter 8: *CCU Payload Communications*.

Telemetry Packet Format

The Level 2 Telemetry packet format is similar to the ACK/NAK packet, but with different data Type values. Also, the Level 1 data has no heading structure.

Figure 7.1. Level 2 Telemetry Data Packet



Type: [1 byte] Telemetry Type (see "Telemetry Types" (page 107), below)

Transaction ID: [4 bytes] This field is not used and is set to **0x00000000**

Size: [2 bytes] The size of the data that follows.

Telemetry Data: [≤ 196 bytes] Telemetry Level 1 data payload.

CRC: [2 bytes] The CRC-16 of all bytes in the data packet (except the two CRC bytes).

Telemetry Types

Currently-defined Telemetry Types are:

Liquid Robotics Proprietary and Confidential

Table 7.1. Telemetry Types

Type	Description	References
0xA6	Encrypted Output Packet	See "Encrypted Output Packet 0xA6" (page 108)
0xB0	Compressed (GZIP) Output Packet	See "Compressed (GZIP) Output Packet" (page 109)
0xB1	Compressed (BZIP2) Output Packet	See "Compressed (BZIP2) Output Packet" (page 109)
0xC0	Multi-L2 Output Packet	See "Multi-L2 Output Packet" (page 109)
0xD	Computed Course Telemetry Type 13	See "Computed Course Telemetry Packet Type 13" (page 110).
0xe	Resilient Tracker	See "Resilient Tracker Telemetry " (page 111).
0x08	DB3.5 Power Control Status	See "DB3.5 Power Control Status 0x08" (page 117)
0x09	Twist Status	See "Twist Status 0x09" (page 121)
0x10	Mobile WGMS Heartbeat	See "Mobile WGMS Heartbeat 0x10" (page 122)
0x11	Mobile WGMS Sync Packet	See "Mobile WGMS Sync Packet 0x11" (page 122)
0x32	Reserved	
0x44	Reserved	
0x70	Relay Status	See "Relay Status 0x70" (page 122)
0x71	System Information	See "System Information 0x71" (page 123)
0x80	Single Payload Packet	See "Single Payload Packet 0x80" (page 125)
0x81	Multi-Payload Packet	See "Multi-Payload Packet 0x81" (page 146)
0x91	Payload Packet	See "Payload Packet 0x91" (page 146)

Telemetry Buffering

As with ACK/NAK responses, Iridium service may occasionally be interrupted for up to several minutes. In this case, the CCU electronics will buffer any telemetry, and then send out buffered messages when service is resumed.

Since the message buffer is 16 responses deep, once the 16-message limit is reached, any additional messages will be added to the buffer, and the oldest message discarded. Thus, for very long Iridium disruptions, some data may be lost.

Encrypted Output Packet 0xA6

Encrypted output is decrypted and re-packaged by the relay into the clear text form (the header size is changed to reflect clear text size).

Encrypted Output packets are transmitted between relays and vehicles in secure mode. A cipher level is implemented to transfer the LEVEL 2 data packet. Type is maintained to identify the direction of the

Liquid Robotics Proprietary and Confidential

packet (from/to vehicle) and counts are required to distinguish between the level 2 cipher data and the AES initialization vector data in the packet. If IV is not included (size = 0) then IV is assumed to be the previously used IV.

Packet Format:

Type (1)	Size IV (1)	Size cipher (2)	IV	**Level 2 Cipher Data**	CRC (2)

Compressed (GZIP) Output Packet

Compressed packets can be transmitted from ship to shore.

Packet format:

Type (1)	Size (2)	**Level 2 Compressed Packet(s)**	CRC (2)

Type = 0xB0 GZIP compression

Note

Compression Packet when applied has to be applied first before encryption.

Compressed (BZIP2) Output Packet

Compressed packets can be transmitted from ship to shore.

Packet format:

Type (1)	Size (2)	**Level 2 Compressed Packet(s)**	CRC (2)

Type = 0xB1 BZIP2 compression (future support)

Note

Compression Packet when applied has to be applied first before encryption.

Multi-L2 Output Packet

In order to pack more data for larger packet sizes (Iridium 9522B and Virtual Relays), a single L3 header can be specified encompassing multiple L2 in sequence.

Packet format:

Type (1)	Size (2)	Count of packets (2)	**Level 2 Packet(s)**	CRC (2)

Type = 0xC0 Multi-L2 Header Type

The header (L3) has two fields that need to reflect the added size of the multi-L2 packets:

Liquid Robotics Proprietary and Confidential

Field Name	Index	Length (bytes)
Protocol_Revision	0	1
Message_Length	1	2
MO_Header_IEI	3	1
…		
MO_Payload IEI	48	1
MO Payload Length	49	2
Interface Level 2 Content	51	Up to 32K byte

Rules around Multi-L2, encryption and compression:

1. Compression will compress all L2 packets and then create a single L2 Compression header around the compressed multi-L2 packet.

2. Encryption May either encrypt a multiple L2 or compressed L2 (based on Step 1) and wrap that with an encryption header.

3. Each L3 header can contain only the following types of L2 packets:

   a. Multi-L2 in clear text and uncompressed

   b. Single compression header as defined in Step 1

   c. Single Encryption header as defined in Step 2

These rules will prevent a random collection and expansion of packets, and limit the processing to a single level parsing.

Note

Multi-Record Ship to Shore has been replaced with single L3 – Multi L2 packet support.

Computed Course Telemetry Packet Type 13

Whenever a Wave Glider SV3 deviates from a programmed course it will publish its calculated course to WGMS with this telemetry. This course will then be rendered by WGMS on the map display.

Field Name	Byte Index	Size	Bit Index
Current Latitude	0	4	
Current Longitude	4	4	
Unix Time Stamp	8	4	
Number of Points (N)	12	1	
Latitude 0	13	4	
Longitude 0	17	4	
…			
Latitude N-1		4	
Longitude N-1		4	

Liquid Robotics Proprietary and Confidential

Resilient Tracker Telemetry

Item	Bytes	Type	Details
Start Byte	1	Byte	Always 0x0E
Status Bits	1	Bit field	LSBit : 1 = up, 0 = inverted Bits 1-7: unused
Unix Time	4	Signed Integer	1 second resolution
Latitude	4	Signed Integer	600,000 = 1 degree
Longitude	4	Signed Integer	600,000 = 1 degree
Battery Voltage	2	Unsigned Integer	Units of 1mV
Iridium Signal	1	Bit field	Bits 0-2: Signal strength, value 0-5 Bits 3-7: Unused
GPS Signal Strength	1	Byte	Maximum GPS strength
GPS HDOP	1	Byte	Resolution = 1/10

NAL Tracker Telemetry 0x03

The NAL tracker is a commercially-available device used to log time and position data, which can then be sent out as Level 1 data. The Type 0x03 NAL tracker data format is:

NAL Tracker Telemetry Field Name	Bytes Used	Byte #	Value [range]	Note
Decade + Hour	1	0	Decade after 2000 | 24 Hour Time	[1]
Minute	1	1	Time Minutes [0 to 59]	
Second	1	2	Time Integral Seconds	[2]
Decimal Seconds	1	3	0.## [0 to 100]	
Year + Month	1	4	Month | Year after decade	[3]
Day	1	5	[0 to 31]	
Latitude	1	6	Signed Integer Degrees [-90 to 90]	
Latitude	1	7	Integer Minutes [0 to 59]	
Longitude	1	8	Unsigned Integer Degrees [0 to 180]	
Longitude	1	9	Integer Minutes [0 to 59]	
Latitude	1	10	Fraction Minutes 0.## [0 to 100]	
Latitude	1	11	Fraction Minutes 0.00## [0 to 100]	
Longitude	1	12	Fraction Minutes 0.#	[4]
Longitude	1	13	Fraction Minutes 0.0## [0 to 100]	
Longitude	1	14	Fraction Minutes 0.00## [0 to 100]	
(not used)	13	15-28 except 27		

Liquid Robotics Proprietary and Confidential

NAL Tracker Telemetry Field Name	Bytes Used	Byte #	Value [range]	Note
Flags	1	27		[5]
Configuration Byte	1	29		

Notes:

[1] The Decade (after 2000) and the Hour are encoded into a single Byte as follows:

Decade = Byte modulo

10 Hour = Byte / 10

[2] Integral Time Seconds area encoded into a single Byte as follows:

Seconds = Byte modulo 0x40

[3] The Year and Month are encoded into a single Byte as follows:

Year = Byte modulo 10

Month = Byte / 10

[4] The decimal fraction of longitude is encoded as follows:

0.# = Byte modulo 10

[5] Byte 27 is a flags byte:

Bit 6 (0x20 mask) is 0 for the eastern hemisphere and 1 for the western hemisphere. For example, a longitude reading of 155 west longitude is encoded as 0x9B in byte 8 (an unsigned value), however, the 0x20 bit in byte 27 is non zero so the reading comes out as -155.

Bit 7 (0x40 mask) is 0 if the location reading is not valid, and 1 if the location reading is valid. If byte 27, bit 7 is 0, the location reading telemetry is not valid. If byte 27, bit 7 is 1, the location reading telemetry is valid.

Shout Tracker Telemetry 0x05

Byte 0 = 0x05 (format type)

Byte 1 is a bit map of the input pins (read as 0)

Bytes 2--9 is a 64 bit unsigned integer -- convert it to the decimal value and then use DIV and MOD type functions to pull out the appropriate digits based on the dLLLMMFFFllmmfffHHHH format where:

d = latitude direction (0 North, 1 South)
l = latitude degrees
m = latitude minutes
f = latitude fraction minutes
L = longitude degrees with direction (<200 East, >200 West)
M = longitude minutes
F = longitude fraction minutes
H = horizontal dilution of precision which is set to always have 4 digits with the last 2 being decimal values

Liquid Robotics Proprietary and Confidential

Bytes 10--17 is a 64 bit unsigned integer -- treat it as a decimal value and then use DIV and MOD type functions to pull out the appropriate digits based on the sMMDDYYYYHHmmSSTVVVV format where:

s = vertical velocity sign (1 positive, 0 negative) (Not Recorded)
MM = month of year
DD = day of month
YYYY = year
HH = hour
mm = minute of hour
SS = seconds
T = tenths of seconds (hundredths of second is always a 0 because of time alignment
VVVV = Last 4 digits in vertical velocity

Bytes 18--25 is a 64 bit unsigned integer -- treat it as a decimal value and then use DIV and MOD type functions to pull out the appropriate digits based on the sCCCCCaAAAAAgGGGGGOO format where:

s = altitude sign (1 positive, 0 negative) – Not Recorded
CCCCC = course in degrees with always 2 digits behind the decimal
a = altitude decimal place – Not Recorded
AAAAA = altitude value – Not Recorded
g = ground velocity decimal place
GGGGG = ground velocity value
OO = 6 output pins. This number can range from 00 to 63 and each bit represents one pin value. (Not Recorded)

Byte 26--27 is a 16 bit unsigned integer -- treat it as a decimal value and then use DIV and MOD type functions to pull out the appropriate digits based on the DVVVV format where:

D = The first digit is the decimal place in the vertical velocity.
VVVV = The next 4 digits are the VDOP value which is set to always have 4 digits with the last 2 being decimal values. (Not Recorded)

Byte 28 is an unsigned 8 bit value -- treat it as a decimal value and then use DIV and MOD type functions to pull out the appropriate digits based on the SSV format where:

SS = The first 2 digits are the number of GPS satellites used. (Not Recorded)
V = The last digit is the first digit in the vertical velocity. (Not Recorded)

Byte 29 is the GPS Quality Indicators and is decoded according to the following table:

Bit	Value
0	2D-Fix (Not Recorded)
1	Dead Reckoning (Not Recorded)
2	Valid Position Fix (Not Recorded)
3	ID Included or extra data follows (Not Recorded)
4	Emergency Alert (Not Recorded)
5	Motion sensor triggered report, 1 if triggered by motion sensor. (Not Recorded)

Liquid Robotics Proprietary and Confidential

Bit	Value
6	Unused
7	Unused

Standard Telemetry Data 0x06

This is the standard telemetry package sent at the defined intervals. The Type 0x06 Telemetry format is:

Field Name	Byte Index	Size	Bit Index
Sub Leak Alarm	0	1	0
Float Leak Alarm	0	1	1
Sub to Float Comms Alarm	0	1	2
Payload Error Condition Alarm	0	1	3
Float Battery Low Alarm	0	1	4
Umbilical Fault Alarm	0	1	5
Sub Pressure Threshold Alarm	0	1	6
GPS Not Functioning	0	1	7
Internal vehicle Comm Error	1	1	0
Sub Reboot Alarm	1	1	1
Float to Sub Comms Error	1	1	2
Over Current Error	1	1	3
Float Temperature Threshold Exceeded	1	1	4
Float Pressure Threshold Exceeded	1	1	5
Sub Temperature Threshold Exceeded	1	1	6
Float Reboot Alarm	1	1	7
Temp_Surface	2	2	
Latitude	4	4	
Longitude	8	4	
Last_Fix_Time_Hours	12	1	
Last_Fix_Time_Minutes	13	1	
Last_Fix_Time_Seconds	14	1	
Last_Fix_Date_Month	15	1	
Last_Fix_Date_Day	16	1	
Last_Fix_Date_Year	17	1	
File Read Transaction Timeout Error	18	1	0
File Write Fallback Image Activated	18	1	1
File Write Program Image Burned	18	1	2
File Write Fallback Image Bad or Not Found	18	1	3
File Write Selected Image Bad or Not Found	18	1	4
File Write Tentative Image Retry	18	1	5

Liquid Robotics Proprietary and Confidential

Field Name	Byte Index	Size	Bit Index
Boot Config Corrupt **(SV2 Only)**	18	1	6
IPIB Not Responding **(SV2 Only)**	18	1	7
Temperature_Sub	19	2	
Distance_Over_Ground (m)	21	2	
Rudder Count	23	4	
Heading_Sub (degrees)	27	2	
Iridium_Signal_Strength	29	1	
Target_WayPoint	30	1	
Desired_Bearing (Degrees)	31	2	
Heading_Float (Degrees)	33	2	
Pressure_Sensor_Float	35	1	
Pressure_Sensor_Sub	36	1	
Navigation_Mode	37	1	See Nav Modes
Light_State	38	1	0
IR_State	38	1	1
Xbee_State **(SV2 Only)**	38	1	2
Payload 1 Power	38	1	3
Payload 2 Power	38	1	4
Sub Payload Power	38	1	5
Bubble Com Error **(SV2 Only)**	38	1	6
Battery Com Error	38	1	7
Reset_Flags	39	1	
Total_Power (10mWh)	40	2	
Iridium_Read_Errors	42	1	
Iridium_Write_Errors	43	1	
Iridium_Session_Errors	44	1	
Water Speed	45	2	
Water Direction	47	2	
Water Speed Std Dev	49	1	
Water Direction Std Dev	50	1	
Humidity **(SV2 Only)**	51	1	
Payload 3 Status	52	1	0
Payload 4 Status	52	1	1
PEP Status **(SV2 Only)**	52	1	2
IPIB Status **(SV2 Only)**	52	1	3
Modem Power	52	1	4
Increased Mailbox Checks	52	1	5

Liquid Robotics Proprietary and Confidential

Field Name	Byte Index	Size	Bit Index
Reserved	52	1	6
Reserved	52	1	7

Alarm Bytes

For the Alarm bytes 0 & 1, the bit values indicate the following:

0 → OK

1 → Alarm Active

Alarm Type	Byte	Bit
Sub Leak Alarm	0	0
Float Leak Alarm		1
Sub-to-Float Comm Alarm		2
Payload Error Condition Alarm		3
Float Battery Low Alarm		4
Umbilical Fault Alarm		5
Sub Pressure Threshold Alarm		6
GPS Not Functioning		7
Unused	1	0
Sub Reboot Alarm		1
Float-to-Sub Comm Error		2
Unused		3
Float Temperature Threshold Exceeded		4
Float Pressure Threshold Exceeded		5
Sub Temperature Threshold Exceeded		6
Float Reboot Alarm		7

Float Status Byte

For the Float Status byte 38, the bit values indicate the following:

0 → Off

1 → On

Status Type	Byte	Bit
Light State	38	0
IR State		1
XBee State **(SV2 Only)**		2
Payload 1 Power		3
Payload 2 Power		4
Unused		5

Liquid Robotics Proprietary and Confidential

Status Type	Byte	Bit
Unused		6
Unused		7
Payload 3 Status	52	0
Payload 4 Status		1
PEP Status **(SV2 Only)**		2
IPIB Status **(SV2 Only)**		3

Telemetry - Power Controller Status 0x7

A condensed format for the Power Controller Status is shown in the table below. The complete description is given in "ACK - Power Controller Status (0x14)" (page 99) The Type 0x07 Telemetry format is:

Power Controller Data Type	Bytes Used	Byte #
Hours/Minutes/Seconds	3	0-2
Day/Month/Year	3	3-6
Payload 1 & 2 Currents	2	6-7
Umbilical Current	1	8
Untwist Current	1	9
Total Power Remaining	2	10-11
Battery Current	2	12-13
System Current	2	14-15
System Voltage	2	16-17
Solar Power 1/2/3/4 Indicators	4	18-21
Battery_0 Status	8	22-29
Battery_1 Status	8	30-37
Battery_2 Status	8	38-45
Battery_3 Status	8	46-53
Battery_4 Status	8	54-61
Battery_5 Status	8	62-69
Battery_6 Status	8	70-77

DB3.5 Power Control Status 0x08

Power Control Status packet format:

Field Name	Byte Index	Size	Bit Index	Default
Hour	0	1		
Minute	1	1		
Second	2	1		
Month	3	1		

Liquid Robotics Proprietary and Confidential

Field Name	Byte Index	Size	Bit Index	Default
Day	4	1		
Year	5	1		
Payload_1_Current	6	2		(mA) Signed
Payload_2_Current	8	2		(mA) Signed
Umbilical_Current	10	2		(mA) Signed
Internal_Payload **(SV2 Only)**	12	2		(mA) Signed
Payload_3_Current	14	2		(mA) Signed
Payload_4_Current	16	2		(mA) Signed
Power_Extension_Port	18	2		(mA) Signed
Light_Bar	20	2		(mA) Signed
Total_Power_Remaining	22	4		(mWH)
Battery_Current	26	2		Signed
Payload_Voltage	28	2		(mV)
Float_Voltage	30	2		(mV)
Payload_Current	32	2		(mA)
Float_Current	34	2		(mA)
Modem_Current	36	2		(mA)
Solar_Power_1_Power	38	2		(mW)
Solar_Power_2_Power	40	2		(mW)
Auxiliary_Power	42	2		(mW)
Reserved_Power	44	2		(mW)
PM_Status	46	1		
Reserved_0	47	1		
Reserved_1	48	1		
Reserved_2	49	1		
Reserved_3	50	1		
Reserved_4	51	1		
Reserved_5	52	1		
Reserved_6	53	1		
Reserved_7	54	1		
Reserved_8	55	1		
Reserved_9	56	1		
Reserved_10	57	1		
Reserved_11	58	1		
Reserved_12	59	1		
Count of batteries in report	60	1		0 – 7 batteries >0 – count of batteries

Liquid Robotics Proprietary and Confidential

Field Name	Byte Index	Size	Bit Index	Default
Battery_0_Present	61	1	0	
Battery_0_Discharged	61	1	1	
Battery_0_Charged	61	1	2	
Battery_0_Discharging	61	1	3	
Battery_0_Terminate_Discharge	61	1	4	
Battery_0_Terminate_Charge	61	1	5	
Battery_0_Over_Temp	61	1	6	
Battery_0_Online	61	1	7	
Battery_0_Temp	62	1		
Battery_0_Power_Remaining	63	2		
Battery_0_Current	65	2		Signed
Battery_0_Voltage	67	2		
Battery_1_Present	69	1	0	
Battery_1_Discharged	69	1	1	
Battery_1_Charged	69	1	2	
Battery_1_Discharging	69	1	3	
Battery_1_Terminate_Discharge	69	1	4	
Battery_1_Terminate_Charge	69	1	5	
Battery_1_Over_Temp	69	1	6	
Battery_1_Online	69	1	7	
Battery_1_Temp	70	1		
Battery_1_Power_Remaining	71	2		
Battery_1_Current	73	2		Signed
Battery_1_Voltage	75	2		
Battery_2_Present	77	1	0	
Battery_2_Discharged	77	1	1	
Battery_2_Charged	77	1	2	
Battery_2_Discharging	77	1	3	
Battery_2_Terminate_Discharge	77	1	4	
Battery_2_Terminate_Charge	77	1	5	
Battery_2_Over_Temp	77	1	6	
Battery_2_Online	77	1	7	
Battery_2_Temp	78	1		
Battery_2_Power_Remaining	79	2		
Battery_2_Current	81	2		Signed
Battery_2_Voltage	83	2		
Battery_3_Present	85	1	0	
Battery_3_Discharged	85	1	1	

Liquid Robotics Proprietary and Confidential

Field Name	Byte Index	Size	Bit Index	Default
Battery_3_Charged	85	1	2	
Battery_3_Discharging	85	1	3	
Battery_3_Terminate_Discharge	85	1	4	
Battery_3_Terminate_Charge	85	1	5	
Battery_3_Over_Temp	85	1	6	
Battery_3_Online	85	1	7	
Battery_3_Temp	86	1		
Battery_3_Power_Remaining	87	2		
Battery_3_Current	89	2		Signed
Battery_3_Voltage	91	2		
Battery_4_Present	93	1	0	
Battery_4_Discharged	93	1	1	
Battery_4_Charged	93	1	2	
Battery_4_Discharging	93	1	3	
Battery_4_Terminate_Discharge	93	1	4	
Battery_4_Terminate_Charge	93	1	5	
Battery_4_Over_Temp	93	1	6	
Battery_4_Online	93	1	7	
Battery_4_Temp	94	1		
Battery_4_Power_Remaining	95	2		
Battery_4_Current	97	2		Signed
Battery_4_Voltage	99	2		
Battery_5_Present	101	1	0	
Battery_5_Discharged	101	1	1	
Battery_5_Charged	101	1	2	
Battery_5_Discharging	101	1	3	
Battery_5_Terminate_Discharge	101	1	4	
Battery_5_Terminate_Charge	101	1	5	
Battery_5_Over_Temp	101	1	6	
Battery_5_Online	101	1	7	
Battery_5_Temp	102	1		
Battery_5_Power_Remaining	103	2		
Battery_5_Current	105	2		Signed
Battery_5_Voltage	107	2		
Battery_6_Present	109	1	0	
Battery_6_Discharged	109	1	1	
Battery_6_Charged	109	1	2	
Battery_6_Discharging	109	1	3	

Liquid Robotics Proprietary and Confidential

Field Name	Byte Index	Size	Bit Index	Default
Battery_6_Terminate_Discharge	109	1	4	
Battery_6_Terminate_Charge	109	1	5	
Battery_6_Over_Temp	109	1	6	
Battery_6_Online	109	1	7	
Battery_6_Temp	110	1		
Battery_6_Power_Remaining	111	2		
Battery_6_Current	113	2		Signed
Battery_6_Voltage	115	2		

Twist Status 0x09

Twist Status packet format:

Field Name	Byte Index	Size	Bit Index	Default
Hour	0	1		
Minute	1	1		
Second	2	1		
Month	3	1		
Day	4	1		
Year	5	1		
Motor_Activity	6	1	0, 1	
Twist_Direction	6	1	2	
Not_Used_3	6	1	3	
Twist_Activity	6	1	4	
Twist_Configuration	6	1	5	
Twist_Calibration_0	6	1	6	
Twist_Calibration_1	6	1	7	
Not_Used_8	7	1	0	
Not_Used_9	7	1	1	
Not_Used_10	7	1	2	
Not_Used_11	7	1	3	
Not_Used_12	7	1	4	
Not_Used_13	7	1	5	
Not_Used_14	7	1	6	
Twist_Debug	7	1	7	
Twist_Speed	8	2		(RPM)
Twist_Location	10	2		0-4795 = 0-360

Field Name	Byte Index	Size	Bit Index	Default
Number_Of_Position	12	2		signed
Gearing_Ration	14	2		
Calibration_Offset	16	2		
Calibration_Threshold_A	18	2		Signed
Calibration_Threshold_B	20	2		Signed
Home_Position_A	22	2		Signed
Max_Delta_A	24	2		Signed
Home_Position_B	26	2		Signed
Max_Delta_B	28	2		Signed
MaxEffect_A	30	2		Signed
Max_Delta_Home_A	32	2		Signed
MaxEffect_B	34	2		Signed
MaxDelta_B	36	2		Signed

Mobile WGMS Heartbeat 0x10

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
Disposition	1	12	Extended Channel Status Byte
Mobile WGMS Id	10	13 – 22	Mobile WGMS Id

Mobile WGMS Sync Packet 0x11

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
Mobile WGMS Id	10	12 – 21	Mobile WGMS Id
Command Level 3	Variable	22 –	

Relay Status 0x70

Level 1 Format:

Field Name	Byte Index	Size	Bit Index	Default
Channel State	0	1	0	0=not active

Liquid Robotics Proprietary and Confidential

Field Name	Byte Index	Size	Bit Index	Default
				1=active
Message Not Sent Alarm	0	1	1	
Communications Error Alarm	0	1	2	
Channel Not Authenticated Alarm	0	1	3	
Unused	0	1	4	0
Unused	0	1	5	0
Unused	0	1	6	0
Channel Encrypted Status	0	1	7	0=not active 1= active
Channel Revision major	1	1		2
Channel Revision Minor	2	1		0
Channel Revision Sub-Minor	3	1		0
Hour*	4	1		
Minute*	5	1		
Second*	6	1		
Month*	7	1		
Day*	8	1		
Year – 2000*	9	1		
Signal Strength**	10	1		0 – 5 bars

*Timestamp Date-Time in UTC.

** Only present in Virtual Relay (Structure Type >= 32)

Virtual Relay Channel Status Handshake

This 5 byte handshake between remote applications and the Virtual Relay in response to a RelayStatus, is sent prior to sending any pending commands in response to a connection.

Field Name	Bytes	Default Value
Relay Status Size	2	
Relay Status CRC16	2	
Disposition	1	0 – Channel Not Secure 1 – Channel is Secure

System Information 0x71

This message format is identical to that of Message type 0x91 except that the information contained in this message is a system information/configuration in nature.

Field Name	Bytes Used	Byte #	Format
System Information Type	4	0-3	

Liquid Robotics Proprietary and Confidential

Field Name	Bytes Used	Byte #	Format
Board Id	1	4	Unsigned 8-bit integer
Task Id	1	5	Unsigned 8-bit integer
Data Format	1	6 (bits 0-3)	Unsigned 4-bit integer
Default Parser	1	6 (bit 4)	0=Binary/1=ASCII
Delivery Priority	1	6 (bits 5-7)	Unsigned 3-bit integer
Repeat Count	1	7	Unsigned 8-bit integer
System Information Record(s)	[variable]	8	

System Information Records

(Telemetry Type 113) All system information records must provide space and time using the following format. Message 0x71 provides for repeat counts for multiple samples within a message.

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
System Information Data	[variable]	12-	

System Information Types

These are the different types of system information packets sent from the vehicle.

Field Name	Number
Unspecified	0
Partition State	1
Event Messages	2
Crypto Chip Statuses	3
AMPS J Power	4
Event (ver 2) Messages	5
Mobile WGMS Heartbeat	10
Mobile WGMS Sync	11
DB3.5 Acc Power Status	32

Partition State Record (Type 1)

The partition state record indicates the state of the encrypted partition on the System Management Computer (SMC).

Field Name	Bytes Used	Byte #	Format
Latitude	4	0-3	Signed 32-bit integer
Longitude	4	4-7	Signed 32-bit integer

Liquid Robotics Proprietary and Confidential

Field Name	Bytes Used	Byte #	Format
Unix Timestamp	4	8-11	Signed 32-bit integer
Partition State	1	12	0: partition disabled 1: partition enabled

Single Payload Packet 0x80

This telemetry type sends data packets that originate from the vehicle payload section. The Payload Packet data is given a type-indicating header by the CCU, then sent to remote applications. There is no checking or processing of the payload data by the CCU.

Field Name	Bytes
Packet Type Value	4
Packet Data	[variable]

The first four bytes of the packet indicate the data type. The Payload Telemetry types are:

Packet Type	Value	Reference
Weather Station	1	“Weather Station Packet (Type 1)” (page 125)
Payload Status	2	“Payload Status Packet (Type 2)” (page 126)
Payload AIS Status	16	“Payload AIS Status Packet (Type 16)” (page 127)
Payload AIS Static Record	17	“Payload AIS Static Record (Type 17)” (page 128)
Payload AIS Position Report	18	“Payload AIS Position Report (Type 18)” (page 129)

Weather Station Packet (Type 1)

A Type 1 Weather Station data packet has 35 bytes and contains the following data:

Field Name	Bytes Used	Byte #	Value (units)	Notes
Weather Data Version	2	0-1	(internal)	
Flags	2	2-3	(internal)	
Temperature	2	4-5	°C / 10	
Pressure	2	6-7	(millibars)	
Latitude	4	8-11	(± minutes / 10,000)	[1]
Longitude	4	12-15	(± minutes / 10,000)	[2]
Avg. Wind Speed	2	16-17	(knots / 10)	
Max Wind Speed	2	18-19	(knots / 10)	
Std. Dev. Wind Speed	2	20-21	(knots / 10)	
Avg. Wind Direction	2	22-23	(degrees / 10)	

Liquid Robotics Proprietary and Confidential

Field Name	Bytes Used	Byte #	Value (units)	Notes
Std. Dev. Wind Dir	2	24-25	(degrees / 10)	
Number of Wind Samples	2	26-27	(count)	
Hour	1	28	[0 – 23] (UTC)	
Minute	1	29	[0 – 59]	
Second	1	30	[0 – 59]	
Day	1	31	[1 – 31]	
Month	1	32	[1 – 12]	
Year	2	33-34	[2000 – 2099]	

Notes:

[1] Latitude: (+) → North; (–) → South

[2] Longitude: (+) → East; (–) → West

Payload Status Packet (Type 2)

A Type 2 Payload Status packet has 16 bytes and contains the following data:

Field Name	Bytes Used	Byte #	Bit #	Value (units)
Hour	1	0		
Minute	1	1		
Second	1	2		
Month	1	3		
Day	1	4		
Year	1	5		
Version	2	6		
Alarm0 – Leak1	1	8	0	**0** → OK, **1** → Alarm
Alarm1 – Leak2			1	**0** → OK, **1** → Alarm
Alarm2 – Pressure Low			2	**0** → OK, **1** → Alarm
Alarm3 – Pressure High			3	**0** → OK, **1** → Alarm
Alarm4 – Humidity **(SV2 Only)**			4	**0** → OK, **1** → Alarm
Alarm5 – Temp Humid			5	**0** → OK, **1** → Alarm
Alarm6 – Temp Pressure			6	**0** → OK, **1** → Alarm
Alarm7			7	
Alarm8	1	9	0	
Alarm9			1	
Alarm10			2	
Alarm11			3	
Alarm12			4	

Liquid Robotics Proprietary and Confidential

Field Name	Bytes Used	Byte #	Bit #	Value (units)
Alarm13			5	
Alarm14			6	
Alarm15			7	
Leak1	1	10		
Leak2	1	11		
HumidTemp	1	12		(°C)
Humidity **(SV2 Only)**	1	13		(%)
PressureTemp	1	14		(°C)
Pressure	1	15		(kPa)

Payload Enhanced Pressure Weather Station Report Type 5

Field Name	Byte Index	Size	Bit Index	Units
Weather Data Version	0	2		
Flags	2	2		
Temperature	4	2		1/10 deg c
Pressure	6	2		1/10 mBars
Latitude	8	4		
Longitude	12	4		
Avg. Wind Speed	16	2		1/10 knots
Max Wind Speed	18	2		1/10 knots
Std. Dev. Wind Speed	20	2		1/10 knots
Avg. Wind Direction	22	2		1/10 deg
Std. Dev. Wind Dir	24	2		1/10 deg
Number of Wind Samples	26	2		
Hour	28	1		
Minute	29	1		
Second	30	1		
Day	31	1		
Month	32	1		
Year	33	2		

Payload AIS Status Packet (Type 16)

A Payload AIS Status packet has 16 bytes and contains the following data:

Liquid Robotics Proprietary and Confidential

Field Name	Bytes Used	Byte #	Value (units)
Hour	1	0	
Minute	1	1	
Second	1	2	
Month	1	3	
Day	1	4	
Year	1	5	
Latitude	4	6	
Longitude	4	10	
Temperature	2	14	
Pressure	2	16	
Humidity **(SV2 Only)**	2	18	
Alarms	2	20	(TBD)

Payload AIS Static Record (Type 17)

A Payload AIS Static Record packet has 86 bytes and contains the following data:

Field Name	Bytes Used	Byte #	Default
Hour	1	0	
Minute	1	1	
Second	1	2	
Month	1	3	
Day	1	4	
Year	1	5	
MMSI	4	6	
AISClass	1	10	A/B
Ship Name (20 characters)	20	11	
Call Sign (7 characters)	7	31	
Ship and Cargo Type	1	38	
Device Used for the Position Fixes	1	39	
Destination (20 characters)	20	40	
ETA (Month, Day, Hour, Minutes)	4	60	
Dimension and Point Of Reference	4	64	
IMO	4	68	
Latitude	4	72	
Longitude	4	76	
Course Over Ground	2	78	
Speed Over Ground	2	80	
Rate Of Turn	1	82	

Liquid Robotics Proprietary and Confidential

Field Name	Bytes Used	Byte #	Default
Heading	2	83	
Navigation Status	1	85	

Payload AIS Position Report (Type 18)

A Payload AIS Static Record packet has a variable number of bytes depending on the number of records contained. It has the following format:

Field Name	Bytes Used	Byte #	Value
Number Of Records in Message	1	0	**N**
Record 1	26	1	(see record format below)
Record 2	26	27	(see record format below)
. . .	. . .	. . .	
. . .	. . .	. . .	
Record N	26		(see record format below)

Each 26-byte Record has the following format:

Field Name	Bytes Used	Byte #	Default
Hour	1	0	
Minute	1	1	
Second	1	2	
Month	1	3	
Day	1	4	
Year	1	5	
MMSI	4	6	
Latitude	4	10	
Longitude	4	14	
Course Over Ground	2	18	
Speed Over Ground	2	20	
Rate Of Turn	1	22	
Heading	2	23	
Navigation Status	1	25	

Sensor Sample Record (Unparsed)

All sensors must locate each sample in space and time using the following format. The software managing the sensor is responsible for providing the timestamp and the lat/lon. The SMC (and Payload Interface Boards (PIBs) - these are SV2 Sensor Port Payloads adapted for SV3) should provide callable

Liquid Robotics Proprietary and Confidential

routines that report these values in a form that can be directly inserted into sensor sample records. Message types 0x90 and 0x91 provide for repeat counts for multiple samples within a message.

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
Sensor Data	[variable]	12-	

Fluorometer (C3) Data Type 96

Data Format 0:

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
DataC1 Int	2	12-13	Unsigned 32-bit integer
DataC1 Fraction	1	14	Unsigned 32-bit integer
DataC2 Int	2	15-16	Unsigned 32-bit integer
DataC2 Fraction	1	17	Unsigned 32-bit integer
DataC3 Int	2	18-19	Unsigned 32-bit integer
DataC3 Fraction	1	20	Unsigned 32-bit integer
Pressure Int	2	21-22	Unsigned 32-bit integer
Pressure Fraction	1	23	Unsigned 32-bit integer
Temperature	2	24-25	Unsigned 32-bit integer

Compressed CTD Data Type 112

The GPCTD Data comes back as payload type 0x00000070 or 112 decimal. The format does use the sub-type field for a couple of different possible data formats. Currently we are only supporting sub-type 8 which is the CTD Data with no O2 data included (details are in section 9 in the document). At some point in the future we will also have units that use sub-type 9, which will have an extra 4 bytes in every GPCTD sample for the oxygen.

Compressed CTD Data

This record reports the Pressure, Temperature, and Conductivity for each sample.

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
Pressure	4	12-15	Unsigned 32-bit integer
Temperature	4	16-19	Unsigned 32-bit integer

Liquid Robotics Proprietary and Confidential

Field Name	Bytes Used	Byte #	Format
Conductivity	4	20-23	Unsigned 32-bit integer

Latitude and Longitude – the lat/lon reported by the CCU to the Payload Interface Board (PIB) of a Sensor Port Payload – the location of the float at the time that the PIB obtained it from the CCU. Note that the reading of the GPS is not directly coordinated with the CTD, but should be close enough given other errors in the system.

Unix Timestamp – the standard Unix timestamp computed from the PIB RTC timestamp. The PIB clock is set from the CCU GPS and should be accurate within a second.

Pressure – The Pressure as measured by the CTD. P(in decibars) = (Value/100) - 10

Temperature – The Temperature as measured by the CTD. T(in deg C) = (Value/10,000) – 5

Conductivity – The Conductivity as measured by the CTD. C(in S/m) = (Value/100,000) – 0.05

Compressed CTD With Oxygen Sensor Data

This record reports the Pressure, Temperature, and Conductivity for each sample.

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
Pressure	4	12-15	Unsigned 32-bit integer
Temperature	4	16-19	Unsigned 32-bit integer
Conductivity	4	20-23	Unsigned 32-bit integer
Oxygen	4	24-27	Unsigned 32-bit integer

Latitude and Longitude – the lat/lon reported by the CCU to the Payload Interface Board (PIB) of a Sensor Port Payload – the location of the float at the time that the PIB obtained it from the CCU. Note that the reading of the GPS is not directly coordinated with the CTD, but should be close enough given other errors in the system.

Unix Timestamp – the standard Unix timestamp computed from the PIB RTC timestamp. The PIB clock is set from the CCU GPS and should be accurate within a second.

Pressure – The Pressure as measured by the CTD. P(in decibars) = (Value/100) - 10

Temperature – The Temperature as measured by the CTD. T(in deg C) = (Value/10,000) – 5

Conductivity – The Conductivity as measured by the CTD. C(in S/m) = (Value/100,000) – 0.05

Oxygen – The Dissolved Oxygen as measured by the optional Oxygen sensor. O(in Hz) = (Value/10)

Payload ECO Puck Sensor Data Type 123

ECO Puck telemetry is returned as a type 0x91 structure as follows. See WET Labs documentation for definitions of the sensor specific values. Where appropriate, the text strings are converted to binary to save space.

Liquid Robotics Proprietary and Confidential

FORMAT CODE 0 - ECO PUCK SAMPLE

Field Name	Bytes Used	Byte #	Format
PayloadType	4	0-3	123 (WET Labs ECO Puck)
Board Id	1	4	Unsigned 8-bit integer
Task Id	1	5	Unsigned 8-bit integer
Sensor Data Format	1	6 (bits 0-3)	Unsigned 4-bit integer – 0
Default Parser	1	6 (bit 4)	0=Binary/1=ASCII – 0
Delivery Priority	1	6 (bits 5-7)	Unsigned 3-bit integer – 0
Repeat Count	1	7	Unsigned 8-bit integer

Individual samples data format 0

Latitude	**4**	**0-3**	**Signed 32-bit integer**
Longitude	4	4-7	Signed 32-bit integer
Time	4	8-11	Unix time_t as read on the SMC
Raw date from sensor	8	12-19	String returned by sensor
Raw time from sensor	8	20-27	String returned by sensor
Reference frequency 1	2	28-29	Unsigned 16-bit integer
Reading 1	2	30-31	Unsigned 16-bit integer
Reference frequency 2	2	32-33	Unsigned 16-bit integer
Reading 2	2	34-35	Unsigned 16-bit integer
Reference frequency 3	2	36-37	Unsigned 16-bit integer
Reading 3	2	38-39	Unsigned 16-bit integer
Therm	2	40-41	Unsigned 16-bit integer

Example of a raw sample with 13 data columns of the EcoPuck:

99/99/99 99:99:99 532 4130 650 4130 460 4130 525 2011/06/28 7:05:45.430877 37.410927833333325 -122.00422849999999

How the first 9 columns of the payload should be interpreted:

	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	Column 9
Unit	Raw Date	Raw Time	Sample 1	532nm	Sample2	650nm	Sample3	CDOM	therm
Raw Samples	99/99/99	99:99:99	532	4130	650	4130	460	4130	525

Liquid Robotics Proprietary and Confidential

Type	8char String	8char String	INT16	INT16	INT16	INT16	INT16	INT16	INT16

Payload CStar Sensor Data Type 124

The telemetry is returned as a type 0x91 structure. See WET Labs documentation for the definition of the sensor specific values. Where appropriate, the text strings are converted to binary to save space.

FORMAT CODE 0 – C-STAR SENSOR TELEMETRY

Field Name	Bytes Used	Byte #	Format
PayloadType	4	0-3	124 (WET Labs ECO Puck)
Board Id	1	4	Unsigned 8-bit integer
Task Id	1	5	Unsigned 8-bit integer
Sensor Data Format	1	6 (bits 0-3)	Unsigned 4-bit integer – 0
Default Parser	1	6 (bit 4)	0=Binary/1=ASCII – 0
Delivery Priority	1	6 (bits 5-7)	Unsigned 3-bit integer – 0
Repeat Count	1	7	Unsigned 8-bit integer

Individual samples

Latitude	**4**	**0-3**	**Signed 32 bit integer**
Longitude	4	4-7	Signed 32-bit integer
Unix Timestamp	4	8-11	Unix time_t as read on the SMC
Sensor serial number	10	12-21	String returned by sensor
Reference count	2	22-23	Unsigned 16-bit integer
Signal count	2	24-25	Unsigned 16-bit integer
Corr Signal Raw count	2	26-27	Unsigned 16-bit integer
Beam c Inverse	8	28-35	Double precision floating point
Temperature	2	36-37	Unsigned 16-bit integer

Example of a raw sample with 9 data columns of the CStar:

1234PG 11829 13838 13695 0.003 527 17:05:45.430877
37.410927833333325 -122.00422849999999

How the first 6 columns of the payload should be interpreted:

	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
Unit	Instrument serial number	Reference counts	Signal counts	Corrected Signal raw counts	Calculated Beam c inverse meters	Internal thermistor, counts
Raw Samples	CST-1234PG	11829	13838	13695	0.003	527

Liquid Robotics Proprietary and Confidential

Payload Benthos Modem Data Type 125

Format code 0 – Range telemetry results (ATY/ATR)

Field Name	Bytes Used	Byte #	Format
PayloadType	4	0-3	125 (Benthos Modem)
Board Id	1	4	Unsigned 8-bit integer
Task Id	1	5	Unsigned 8-bit integer
Sensor Data Format	1	6 (bits 0-3)	Unsigned 4-bit integer – 0
Default Parser	1	6 (bit 4)	0=Binary/1=ASCII – 0
Delivery Priority	1	6 (bits 5-7)	Unsigned 3-bit integer – 0
Repeat Count	1	7	Unsigned 8-bit integer – 1
Latitude	4	8-11	Latitude
Longitude	4	12-15	Longitude
Time	4	16-19	Unix time_t
Distance to Target	4	20-23	Unsigned 32-bit integer, tenths of meters

Records of the following format representing results for different baud rates, 6 sets (150 bytes):

Baud Rate	2	0-1	Signed 16-bit integer
Acquisition - SNR	2	2-3	Signed 16-bit integer
Acquisition - MPD	2	4-5	Signed 16-bit integer
Acquisition - SPD	2	6-7	Signed 16-bit integer
Header - CRC	1	8	Signed 8-bit integer (0=fail, 1=pass)
Header - SNR	2	9-10	Signed 16-bit integer
Header – CCERR	2	11-12	Signed 16-bit integer (-1 if not reported)
Data - MOD	2	13-14	Signed 16-bit integer
Data - ERR	2	15-16	Signed 16-bit integer
Data - SNR	2	17-18	Signed 16-bit integer
Data - AGC	2	19-20	Signed 16-bit integer
Data - SPD	2	21-22	Signed 16-bit integer
Data - CCERR	2	23-24	Signed 16-bit integer

Payload Datawell MOSE Sensor Data Type 126

Format code 0 – Reporting Hs/Tav/Tp/Dirp wave spectrum data.

Field Name	Bytes Used	Byte #	Format
PayloadType	4	0-3	126 (Datawell MOSE)
Board Id	1	4	Unsigned 8-bit integer
Task Id	1	5	Unsigned 8-bit integer

Field Name	Bytes Used	Byte #	Format
Sensor Data Format	1	6 (bits 0-3)	Unsigned 4-bit integer – 0
Default Parser	1	6 (bit 4)	0=Binary/1=ASCII – 0
Delivery Priority	1	6 (bits 5-7)	Unsigned 3-bit integer – 0
Repeat Count	1	7	Unsigned 8-bit integer

Records the following format:

Latitude	4	8-11	Latitude
Longitude	4	12-15	Longitude
Time	4	16-19	Unix time_t
Hs	8	20-27	double precision float
Tav	8	28-35	double precision float
Tp	8	36-43	double precision float
Dirp	8	44-51	double precision float
Number of Averaged Spectra	1	52	Signed int
Number of Samples per Spectrum	2	53-54	Signed int
Number of Sample Gaps	2	55-56	Signed int

Payload VEMCO VR2C Data Type 127

Format code 0 – Reporting a single tag reading.

Field Name	Bytes Used	Byte #	Format
PayloadType	4	0-3	127 (VEMCO VR2C)
Board Id	1	4	Unsigned 8-bit integer
Task Id	1	5	Unsigned 8-bit integer
Sensor Data Format	1	6 (bits 0-3)	Unsigned 4-bit integer – 0
Default Parser	1	6 (bit 4)	0=Binary/1=ASCII – 1
Delivery Priority	1	6 (bits 5-7)	Unsigned 3-bit integer – 0
Repeat Count	1	7	Unsigned 8-bit integer
Record Format			
Latitude	4	8-11	Latitude
Longitude	4	12-15	Longitude
Time	4	16-19	Unix time_t
Tag Reading String	Varies	20-	string, variable length

Payload ADCP Data Type 128

Bathymetry Data:

Field Name	Bytes Used	Byte #	Format
Latitude	4	0-3	Signed 32-bit integer

Liquid Robotics Proprietary and Confidential

Field Name	Bytes Used	Byte #	Format
Longitude	4	4-7	Signed 32-bit integer
Unix Timestamp	4	8-11	Signed 32-bit integer
Heading	2	12-13	Unsigned 16-bit integer
Pitch	2	14-15	Signed 16-bit integer
Roll	2	16-17	Signed 16-bit integer
B1	2	18-19	Unsigned 16-bit integer
B2	2	20-21	Unsigned 16-bit integer
B3	2	22-23	Unsigned 16-bit integer
B4	2	24-25	Unsigned 16-bit integer

C2 Data:

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
Bin size	2	12-13	Unsigned 16-bit integer
Blanking distance	2	14-15	Unsigned 16-bit integer
Bin 1, East	2	16-17	Signed 16-bit integer
Bin 1, North	2	18-19	Signed 16-bit integer
Bin 2, East	2	20-21	Signed 16-bit integer
Bin 2, North	2	22-23	Signed 16-bit integer
…	2	24-25	Signed 16-bit integer

C4 Data:

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
Bin size	2	12-13	Unsigned 16-bit integer
Blanking distance	2	14-15	Unsigned 16-bit integer
Bin 1, East	2	16-17	Signed 16-bit integer
Bin 1, North	2	18-19	Signed 16-bit integer
Bin 1, Up	2	20-21	Signed 16-bit integer
Bin 1, Error	2	22-23	Signed 16-bit integer
…	2	24-25	Signed 16-bit integer

Payload Aquatracka Data Type 130

Format code 0 – Reporting a single tag reading.

Liquid Robotics Proprietary and Confidential

Field Name	Bytes Used	Byte #	Format
PayloadType	4	0-3	130 (Aquatracka)
Board Id	1	4	Unsigned 8-bit integer
Task Id	1	5	Unsigned 8-bit integer
Sensor Data Format	1	6 (bits 0-3)	Unsigned 4-bit integer – 0
Default Parser	1	6 (bit 4)	0=Binary/1=ASCII – 1
Delivery Priority	1	6 (bits 5-7)	Unsigned 3-bit integer – 0
Repeat Count	1	7	Unsigned 8-bit integer
Record Format			
Latitude	4	8-11	Latitude
Longitude	4	12-15	Longitude
Time	4	16-19	Unix time_t
Raw Temp. Mean	4	20-23	float
Raw Ref. Mean	4	24-27	float
Raw Sig. Mean	4	28-31	float
Conc. Mean	4	32-35	float
Temp. Mean	4	36-39	float
Raw Temp. Stddev	4	40-63	float
Raw Ref. Stddev	4	44-47	float
Raw Sig. Stddev	4	48-53	float
Conc. Stddev	4	54-57	float
Temp. Stddev	4	58-61	float

Payload WHOI Micro Modem Type 131

Format code 0 – Informational Report.

Field Name	Bytes Used	Byte #	Format
PayloadType	4	0-3	131 (WHOI Micro Modem)
Board Id	1	4	Unsigned 8-bit integer
Task Id	1	5	Unsigned 8-bit integer
Sensor Data Format	1	6 (bits 0-3)	Unsigned 4-bit integer – 0
Default Parser	1	6 (bit 4)	0=Binary/1=ASCII – 1
Delivery Priority	1	6 (bits 5-7)	Unsigned 3-bit integer – 0
Repeat Count	1	7	Unsigned 8-bit integer
Latitude	4	8-11	Latitude
Longitude	4	12-15	Longitude
Time	4	16-19	Unix time_t
Report string	Varies	20-	string, variable length

Liquid Robotics Proprietary and Confidential

Harmony Payload Type 132

Format Code 0 - String Report

Field Name	Bytes Used	Byte #	Format
PayloadType	4	0-3	132 (Harmony)
Board Id	1	4	Unsigned 8-bit integer
Task Id	1	5	Unsigned 8-bit integer
Sensor Data Format	1	6 (bits 0-3)	Unsigned 4-bit integer – 0
Default Parser	1	6 (bit 4)	0=Binary/1=ASCII – 1
Delivery Priority	1	6 (bits 5-7)	Unsigned 3-bit integer – 0
Repeat Count	1	7	Unsigned 8-bit integer
Latitude	4	8-11	Latitude
Longitude	4	12-15	Longitude
Time	4	16-19	Unix time_t
Report string	Varies	20-	string, variable length

Format Code 1 - Binary Report

Field Name	Bytes Used	Byte #	Format
PayloadType	4	0-3	132 (Harmony)
Board Id	1	4	Unsigned 8-bit integer
Task Id	1	5	Unsigned 8-bit integer
Sensor Data Format	1	6 (bits 0-3)	Unsigned 4-bit integer – 1
Default Parser	1	6 (bit 4)	0=Binary/1=ASCII – 0
Delivery Priority	1	6 (bits 5-7)	Unsigned 3-bit integer – 0
Repeat Count	1	7	Unsigned 8-bit integer
Latitude	4	8-11	Latitude
Longitude	4	12-15	Longitude
Time	4	16-19	Unix time_t
Report data	Varies	20-	Bytes, variable length

Cefas OrbComm Payload Type 134

Data Format 0:

Field Name	Bytes Used	Byte #	Format
PayloadType	4	0-3	133 (OrbComm)
Board Id	1	4	Unsigned 8-bit integer
Task Id	1	5	Unsigned 8-bit integer
Sensor Data Format	1	6 (bits 0-3)	Unsigned 4-bit integer – 0

Field Name	Bytes Used	Byte #	Format
Default Parser	1	6 (bit 4)	0=Binary/1=ASCII – 1
Delivery Priority	1	6 (bits 5-7)	Unsigned 3-bit integer – 0
Repeat Count	1	7	Unsigned 8-bit integer
Record Format			
Latitude	4	8-11	Latitude
Longitude	4	12-15	Longitude
Time	4	16-19	Unix time_t
Data	270	20-varies	String

Turbidity Payload Type 135

Format Code 0:

Field Name	Bytes Used	Byte #	Format
PayloadType	4	0-3	135 (LISST)
Board Id	1	4	Unsigned 8-bit integer
Task Id	1	5	Unsigned 8-bit integer
Sensor Data Format	1	6 (bits 0-3)	Unsigned 4-bit integer – 0
Default Parser	1	6 (bit 4)	0=Binary/1=ASCII – 1
Delivery Priority	1	6 (bits 5-7)	Unsigned 3-bit integer – 0
Repeat Count	1	7	Unsigned 8-bit integer
Record Format			
Latitude	4	8-11	Latitude
Longitude	4	12-15	Longitude
Time	4	16-19	Unix time_t
Total Concentration	4	20-23	float
Mean Particle Size	4	24-27	float
Sample Count	4	28-31	Unsigned 32-bit integer
Serial Number	2	32-33	Unsigned 16-bit integer
Name Length	1	34	byte
File Name	Varies	35	ASCII String

Sonardyne Tsunami Sensor Payload Type 138

Format code 0 – Informational Report

Field Name	Bytes Used	Byte #	Format
PayloadType	4	0-3	new value
Board Id	1	4	Unsigned 8-bit integer
Task Id	1	5	Unsigned 8-bit integer

Liquid Robotics Proprietary and Confidential

Field Name	Bytes Used	Byte #	Format
Sensor Data Format	1	6 (bits 0-3)	Unsigned 4-bit integer – 0
Default Parser	1	6 (bit 4)	0=Binary/1=ASCII – 1
Delivery Priority	1	6 (bits 5-7)	Unsigned 3-bit integer – 0
Repeat Count	1	7	Unsigned 8-bit integer
Latitude	4	8-11	Latitude
Longitude	4	12-15	Longitude
Time	4	16-19	Unix time_t
Report string	Varies	20-	string, variable length

Marine Magnetics Magnetometer Type 140

Format code 0 – Informational Report

Field Name	Bytes Used	Byte #	Format
PayloadType	4	0-3	136 (Marine Magnetics Magnetometer)
Board Id	1	4	Unsigned 8-bit integer
Task Id	1	5	Unsigned 8-bit integer
Sensor Data Format	1	6 (bits 0-3)	Unsigned 4-bit integer – 0
Default Parser	1	6 (bit 4)	0=Binary/1=ASCII – 1
Delivery Priority	1	6 (bits 5-7)	Unsigned 3-bit integer – 0
Repeat Count	1	7	Unsigned 8-bit integer
Latitude	4	8-11	Latitude
Longitude	4	12-15	Longitude
Time	4	16-19	Unix time_t
Raw report string	Varies	20-	string, variable length

Format code 1 – Parse the raw string from the Magnetometer into individual fields. Additionally, a filtered Magnetic Field data value may be returned, if configured.

Field Name	Bytes Used	Byte #	Format	Samples
PayloadType	4	0-3	148 (Marine Magnetics Magnetometer)	
Board Id	1	4	Unsigned 8-bit integer	
Task Id	1	5	Unsigned 8-bit integer	
Sensor Data Format	1	6 (bits 0-3)	Unsigned 4-bit integer – 0	

Liquid Robotics Proprietary and Confidential

Field Name	Bytes Used	Byte #	Format	Samples
Default Parser	1	6 (bit 4)	0=Binary/1=ASCII – 1	
Delivery Priority	1	6 (bits 5-7)	Unsigned 3-bit integer – 0	
Repeat Count	1	7	Unsigned 8-bit integer	
Latitude	4	8-11	Latitude	
Longitude	4	12-15	Longitude	
Time	4	16-19	Unix time_t	
Filtered Mag Field (nT)	4	20-23	float	016088.127 (nT)
Mag Date (YY.JJJ)	6	24-29	string	“YY.JJJ”
Mag Time (HH:MM:SS.s)	10	30-39	string	“HH:MM:SS.s”
Magnetic Field (nT)	4	40-43	float	016088.127 (nT)
Signal Strength	1	44	Unsigned 8-bit integer	105
Depth (m)	4	45-48	float	-4938.7 (m)
Leak	1	49	Unsigned 8-bit integer	0
Measurement Time (ms)	2	51-52	Unsigned 16-bit integer	0465 (ms)
Signal Quality	1	53	Unsigned 8-bit integer	79

Payload CO2 Sensor Type 144

Field Name	Bytes Used	Byte #	Format
PayloadType	4	0-3	144 (pCO2)
Board Id	1	4	Unsigned 8-bit integer
Task Id	1	5	Unsigned 8-bit integer
Sensor Data Format	1	6 (bits 0-3)	Unsigned 4-bit integer – 0
Default Parser	1	6 (bit 4)	0=Binary/1=ASCII – 1
Delivery Priority	1	6 (bits 5-7)	Unsigned 3-bit integer – 0
Repeat Count	1	7	Unsigned 8-bit integer
Record Format			
Latitude	4	8-11	Latitude
Longitude	4	12-15	Longitude
Time	4	16-19	Unix time_t
Custom Format	Varies	20-	Customer Specified

Liquid Robotics Proprietary and Confidential

Sentinel Sensor Data Type 146

Data format 0:

Field Name	Bytes Used	Byte #	Format	Samples
PayloadType	4	0-3	146 (Sentinel Contact Report)	146
Board Id	1	4	Unsigned 8-bit integer	
Task Id	1	5	Unsigned 8-bit integer	
Sensor Data Format	1	6 (bits 0-3)	Unsigned 4-bit integer – 0	
Default Parser	1	6 (bit 4)	0=Binary/1=ASCII – 1	
Delivery Priority	1	6 (bits 5-7)	Unsigned 3-bit integer – 0	
Repeat Count	1	7	Unsigned 8-bit integer	
Latitude	4	8-11	Latitude	20.038968
Longitude	4	12-15	Longitude	-155.834385
Time	4	16-19	Unix time_t	2013-11-08T00:35:26.039Z
Band	1	20	Unsigned 8-bit integer 1-8	
Contact Id	1	21	0-255	
Contact Status	1	22	Value = 0 - 3	0 = no contact, 1 = early contact, 2 = holding contact, 3 = losing contact
Contact Confidence	1	23	Value = 0 - 10	0 = 0%, 1 = 10%, 2 = 20%...10=100%
Bearing	4	24-27	Float 0.0 – 359.9 **deg**	356.25 deg
Bearing Rate of Change	4	28-31	Float -359.9 - +359.9 **deg/ min**	-0.5 deg/min
Bearing Tolerance	1	32	-/+5	
Signal to Noise Ratio	4	33-36	Float -359.9 - +359.9 **dB**	Detection Statistic 19.75 dB
Signal to Noise Rate of Change	4	37-40	Float 0.0 – 100.0 **dB/min**	Detection Statistic Rate -0.5 dB/min
Ambient Noise	4	41-44	Float	116.0 dB/uPa/Hz
Signal Band Maximum Distance	1	45	Unsigned 8-bit integer	
MMSI	4	46-49		338149019

Data format 1:

Field Name	Bytes Used	Byte #	Format	Samples
PayloadType	4	0-3	146 (Sentinel Contact Report)	146
Board Id	1	4	Unsigned 8-bit integer	

Liquid Robotics Proprietary and Confidential

Field Name	Bytes Used	Byte #	Format	Samples
Task Id	1	5	Unsigned 8-bit integer	
Sensor Data Format	1	6 (bits 0-3)	Unsigned 4-bit integer – 0	
Default Parser	1	6 (bit 4)	0=Binary/1=ASCII – 1	
Delivery Priority	1	6 (bits 5-7)	Unsigned 3-bit integer – 0	
Repeat Count	1	7	Unsigned 8-bit integer	
Latitude	4	8-11	Latitude	20.038968
Longitude	4	12-15	Longitude	-155.834385
Time	4	16-19	Unix time_t	2013-11-08T00:35:26.039Z
Band	1	20	Unsigned 8-bit integer 1-8	
Contact Id	2	21	0-255	
Contact Status	1	23	Value = 0 - 3	0 = no contact, 1 = early contact, 2 = holding contact, 3 = losing contact
Contact Confidence	1	24	Value = 0 - 10	0 = 0%, 1 = 10%, 2 = 20%...10=100%
Bearing	4	25-28	Float 0.0 – 359.9 **deg**	356.25 deg
Bearing Rate of Change	4	29-32	Float -359.9 - +359.9 **deg/ min**	-0.5 deg/min
Bearing Tolerance	1	33	-/+5	
Signal to Noise Ratio	4	34-37	Float -359.9 - +359.9 **dB**	Detection Statistic 19.75 dB
Signal to Noise Rate of Change	4	38-41	Float 0.0 – 100.0 **dB/min**	Detection Statistic Rate -0.5 dB/min
Ambient Noise	4	42-45	Float	116.0 dB/uPa/Hz
Signal Band Maximum Distance	1	46	Unsigned 8-bit integer	
MMSI	4	47-50		338149019

GPS Wave Sensor Data Type 147

Field Name	Bytes Used	Byte #	Format	Samples
PayloadType	4	0-3	147 (GPS Waves Sensor)	147
Board Id	1	4	Unsigned 8-bit integer	
Task Id	1	5	Unsigned 8-bit integer	
Sensor Data Format	1	6 (bits 0-3)	Unsigned 4-bit integer – 0	
Default Parser	1	6 (bit 4)	0=Binary/1=ASCII – 1	

Field Name	Bytes Used	Byte #	Format	Samples
Delivery Priority	1	6 (bits 5-7)	Unsigned 3-bit integer – 0	
Repeat Count	1	7	Unsigned 8-bit integer	
Latitude	4	8-11	Latitude	20.038968
Longitude	4	12-15	Longitude	-155.834385
Time	4	16-19	Unix time_t	2013-11-08T00:35:26.039Z
Hs (meters	8	20-27	Double	Significant wave height
Tp (sec)	8	28-35	Double	Peak Period
Dp (degrees)	8	36-43	Double	Dominant Direction
Samples	2	44-45	Unsigned int16	Number of samples used
Fs (Hz)	8	46-53	Double	Sampling Frequency
Sample Gaps	2	54-55	Unsigned int16	Number of missing samples
Ta (sec)	8	56-63	Double	Average Period

CSIR Sensor Data Type 148

Format code 0 – Informational Report

Field Name	Bytes Used	Byte #	Format
PayloadType	4	0-3	148 (CSIR Sensor)
Board Id	1	4	Unsigned 8-bit integer
Task Id	1	5	Unsigned 8-bit integer
Sensor Data Format	1	6 (bits 0-3)	Unsigned 4-bit integer – 0
Default Parser	1	6 (bit 4)	0=Binary/1=ASCII – 1
Delivery Priority	1	6 (bits 5-7)	Unsigned 3-bit integer – 0
Repeat Count	1	7	Unsigned 8-bit integer
Latitude	4	8-11	Latitude
Longitude	4	12-15	Longitude
Time	4	16-19	Unix time_t
Raw report string	Varies	20-	string, variable length

MAGI Sensor Type 150

Format code 0 – Informational Report

Field Name	Bytes Used	Byte #	Format
PayloadType	4	0-3	150 (MAGI Sensor)
Board Id	1	4	Unsigned 8-bit integer
Task Id	1	5	Unsigned 8-bit integer

Liquid Robotics Proprietary and Confidential

Field Name	Bytes Used	Byte #	Format
Sensor Data Format	1	6 (bits 0-3)	Unsigned 4-bit integer – 0
Default Parser	1	6 (bit 4)	0=Binary/1=ASCII – 1
Delivery Priority	1	6 (bits 5-7)	Unsigned 3-bit integer – 0
Repeat Count	1	7	Unsigned 8-bit integer
Latitude	4	8-11	Latitude
Longitude	4	12-15	Longitude
Time	4	16-19	Unix time_t
Raw report string	Varies	20-	string, variable length

Format code 1 – MAGI Phytoflash

Field Name	Bytes Used	Byte #	Format
PayloadType	4	0-3	150 (MAGI Sensor)
Board Id	1	4	Unsigned 8-bit integer
Task Id	1	5	Unsigned 8-bit integer
Sensor Data Format	1	6 (bits 0-3)	Unsigned 4-bit integer – 0
Default Parser	1	6 (bit 4)	0=Binary/1=ASCII – 1
Delivery Priority	1	6 (bits 5-7)	Unsigned 3-bit integer – 0
Repeat Count	1	7	Unsigned 8-bit integer
Latitude	4	8-11	Signed 32-bit integer, coordinate * 600,000
Longitude	4	12-15	Signed 32-bit integer, coordinate * 600,000
Time	4	16-19	Unix time_t
Fo	4	20-23	Signed 32-bit integer
Fm	4	24-27	Signed 32-bit integer
Blank	4	28-31	Signed 32-bit integer
Fv	4	32-35	Signed 32-bit integer
Yield	4	36-39	Float

Payload Thermistor Array Data Type 192

Format code 0:

Field Name	Bytes Used	Byte #	Format
PayloadType	4	0-3	131 (WHOI Micro Modem)
Board Id	1	4	Unsigned 8-bit integer
Task Id	1	5	Unsigned 8-bit integer

Field Name	Bytes Used	Byte #	Format
Sensor Data Format	1	6 (bits 0-3)	Unsigned 4-bit integer – 0
Default Parser	1	6 (bit 4)	0=Binary/1=ASCII – 1
Delivery Priority	1	6 (bits 5-7)	Unsigned 3-bit integer – 0
Repeat Count	1	7	Unsigned 8-bit integer
Latitude	4	8-11	Latitude
Longitude	4	12-15	Longitude
Time	4	16-19	Unix time_t
T1	2	20-21	
T2	2	22-23	
T3	2	24-25	
T4	2	26-27	
T5	2	28-29	
T6	2	30-31	

Payload ASTRA GPS Data Type 259

TBD

Multi-Payload Packet 0x81

Sequence of Payload packets sent from ship to shore using Vehicle core Iridium modem.

Field Name	Byte Index	Size	Bit Index	Default
PayloadType	0	4		
PacketUId	4	4		
PacketCount	8	2		
PacketTotal	10	2		
PacketData	12	Variable		

NOTE: Packet UId is a vehicle unique identifier that is assigned to each multi-packet sequence in order to uniquely identify all packets that belong to that sequence.

Payload Packet 0x91

(Telemetry Type 145) This message format is identical to that of Message type 0x90 except that the whole structure has a 4 byte PayloadType prepended. The PayloadType is identical in format and usage to the field of the same name and byte position in the deprecated Telemetry Type 0x80 message. The PacketData field of 0x80 would correspond to everything in 0x91 after the PayloadType field.

Field Name	Bytes Used	Byte #	Format
PayloadType	4	0-3	See Telemetry type 0x80
Board Id	1	4	Unsigned 8-bit integer
Task Id	1	5	Unsigned 8-bit integer

Liquid Robotics Proprietary and Confidential

Field Name	Bytes Used	Byte #	Format
Sensor Data Format	1	6 (bits 0-3)	Unsigned 4-bit integer
Default Parser	1	6 (bit 4)	0=Binary/1=ASCII
Delivery Priority	1	6 (bits 5-7)	Unsigned 3-bit integer
Repeat Count	1	7	Unsigned 8-bit integer
Sensor Sample(s)	[variable]	8-	

Sensor Sample Records

(Telemetry Type 145) All sensors must locate each sample in space and time using the following format. The software managing the sensor is responsible for providing the time stamp and the latitude/longitude. The System Management Computer (SMC) and Payload Interface Boards (PIBs) of Sensor Port Payloads should provide callable routines that report these values in a form that can be directly inserted into sensor sample records. Message types 0x90 and 0x91 provide for repeat counts for multiple samples within a message.

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
Sensor Data	[variable]	12-	

Chapter 8. CCU Payload Communications

Customers using application-specific electronics on board a Wave Glider need to be able to interact with other resources on the vehicle as well as with remote applications. Liquid Robotics has developed the common generic communications protocols described below. Using these protocols allow Liquid Robotics and its customers to work independently on a project, yet retain communication interoperability.

The CCU Expansion ports support payload communications with a REST interface that has commands encoded as URL parameters - similar to a web application.

The CCU Sensor ports support the SV2 payload communications protocol described in"Sensor Port Payload Communications Protocol" (page 148)

Sensor Port Payload Communications Protocol

The payload communications protocol for the SV3 CCU Sensor ports is the same payload communications protocol that is used on SV2 vehicles.

Note

The Payload Communications Protocol carries vehicle commands and responses (Level 2 messages). For more information, see Appendix A: *Command Descriptions*.

The CCU is able to receive communications from payloads. The CCU action will depend on how the communication is addressed:

- If addressed to the CCU payload message handler, the CCU will deliver the message to the specified address, or send a NAK if it is unable to deliver the message.
- If addressed to the CCU command processor, the CCU will process the command.
- If addressed to another payload, the CCU will route the message to that payload.
- If addressed to the Iridium system, the CCU will send it out in Iridium format.
- If addressed to the cellular network, the CCU will send it out in cellular format (then the Virtual Relay will convert it to Packet Reflector-compatible packets).
- If addressed to WiFi, the CCU will send it out in WiFi format.

Communication Method

Payload messaging relies on serial communications to transmit data packets, using polling to initiate communications and queues to accommodate messages between polling intervals.

Payload Communications Requirements

The communications protocol must allow the CCU, a Payload Interface Board (PIB), or a Sensor Port connected device to:

Liquid Robotics Proprietary and Confidential

- Identify itself and its software version number.
- Be updated with its current location according to GPS.
- Subscribe to various messages that are sent to remote applications.
- Issue commands directly to the Float CCU.
- Be updated with a new program, if the application running on the payload supports it.
- Receive a file from a remote application or send a requested file to a remote application.
- Indicate when there is a new file available to be read.
- Read or store data in the Float CCU file system.
- Be notified when it is about to be powered off.
- Request a power cycle.
- Interact with a Debug Console.

Serial Communication Parameters

Communications between the CCU and the Sensor Port connected devices are:

- 115,200 bps at RS-232 levels
- 8 data bits
- 1 stop bit
- No parity

Communication is throttled using the protocol, so handshaking features like X-On/X-Off, RTS/CTS and DTR/DSR are disabled.

Communications between a Payload Interface Board (PIB) and its connected devices can use other parameters, depending on the kind of PM used. However, Liquid Robotics recommends using the values listed above.

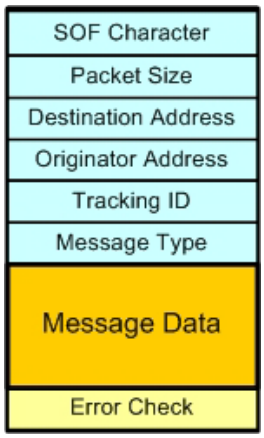
Payload Data Packet Format

Payload data packets contain header data, followed by the message and ending with a CRC error check code. A special 1-byte Start-of-Frame (SOF) character is used to indicate the beginning of each packet ("frame").

The general format is:

Liquid Robotics Proprietary and Confidential

Figure 8.1. Payload Data Packet Format



where:

- **SOF Character:** Indicates the start of a payload message packet.
- **Packet Size:** Is the number of bytes in the packet, excluding the SOF character.
- **Destination Address:** Is where the packet is to be sent.
- **Originator Address:** Is who sent the packet.
- **Tracking ID:** Is the number created by the originator for message tracking.
- **Message Type:** Is a code value indicating the type/format of data in the message.
- **Message Data:** The actual message.
- **Error Check:** A CRC code value.

Message Routing

A message packet contains the address of the originator and the address of the recipient. When the CCU or a Payload Interface Board (PIB) creates a message packet, it can choose to broadcast it to all connected child devices (PIBs, payload boards or applications), or to send it to a specific device. Depending on the message destination ID and the PIB address, the PIB can process the message or route the message to any of its child devices.

Note that a child device cannot send an unsolicited message to its parent device. Thus, a PIB cannot originate a message to the CCU. Likewise, a daisy-chained PIB cannot originate a message to its parent PIB.

When a child device has a packet to send to its parent, it puts the packet in a message queue. The child then has to wait for the parent to poll it. When polled, the PIB responds with a packet that contains the message.

Polling

Certain types of messages have a polling function in addition to any data they may contain.

Liquid Robotics Proprietary and Confidential

The CCU sends out periodic polling messages to the Payload Interface Boards (PIBs) connected to it. The PIBs can send a "message waiting" response to the CCU. If a PIB has a message waiting, the CCU requests the PIB to send it and the PIB sends the message to the CCU. When the CCU receives the message, it routes it to the designated recipient, which can be the CCU itself, another connected PIB, or an external communications device such as Iridium or Cellular.

Similarly, a PIB has the option to send out periodic polling messages to its child devices. Each child device can send a "message waiting" response to the parent PIB. The parent PIB then requests and receives the message, processes the message, and routes it to another child device (or puts it into its message queue and waits to be polled).

The polling rates are set by each PIB as part of the Enumeration Response.

Enumeration Request and Response

At power-up, a specific "Enumeration" message is broadcast by the CCU. In response, each Payload Interface Board (PIB) reports its addresses, along with flags indicating the type of information it would like to receive (including the polling rate). Each PIB then broadcasts its own message to connected child devices, which respond with their own message requirements.

These addresses and flags (along with their physical connectivity) form the basis of the routing tables maintained by the CCU and PIBs.

ACKs and NAKs

When the CCU or a Payload Interface Board (PIB) receives a message packet, it typically responds to the sender with an ACK ("Acknowledge") or NAK ("Not Acknowledge") message packet.

An ACK message indicates that the message was received and was able to be interpreted. When a PIB sends an ACK response to the CCU or parent PIB, it can also indicate that it has a message queued for sending.

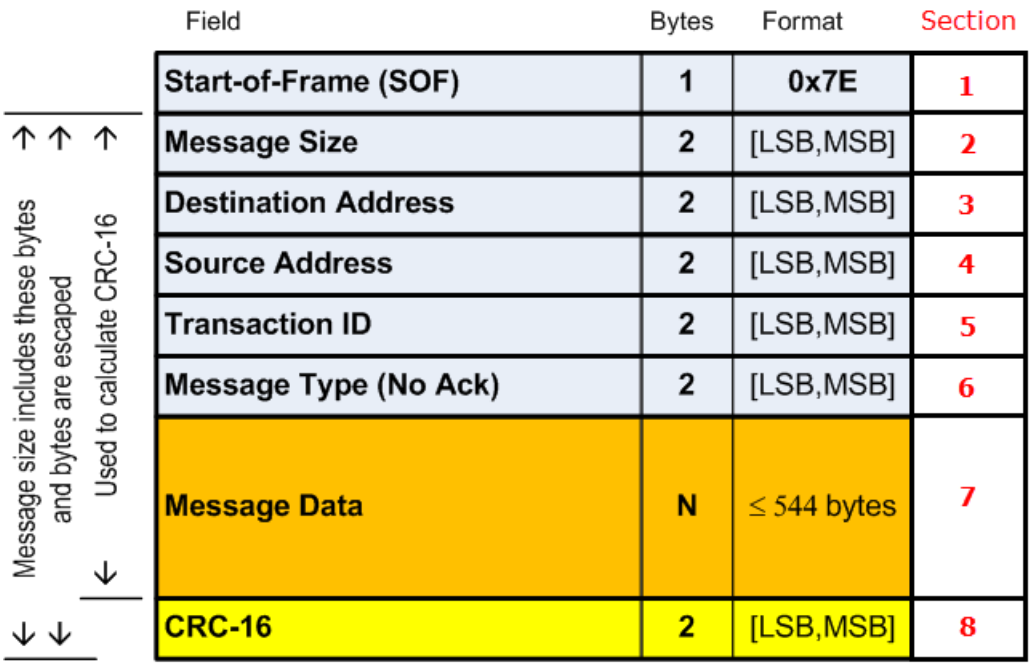
A NAK message indicates that there was a problem with the message. As with the ACK response, a PIB can also use a NAK response to indicate that it has a message queued.

The original message sender has the option of notifying the recipient (using a flag bit) that it doesn't want an ACK/NAK response to the message.

Protocol Message Format

Figure 8.2 shows the format of the message packet for a message containing “N” bytes. Note that all fields are sent in “little-endian” form (LSB first).

Figure 8.2. Message Packet Format



Start-of-Frame Character (Section 1)

The first character in a Message Packet must be the Start-of-Frame (SOF) character: **0x7E**

The SOF character can be escaped to allow it to be embedded within the remainder of the message packet (Sections 2-8).

Message Size (Section 2)

The number of bytes following the Start-of-Frame, including the CRC bytes. This must be **≤ 556 bytes**, however, the limit does not include any added Escape (ESC) bytes.

Destination Address (Section 3)

This field indicates the intended recipient of the message by address. There is also a special reserved address for broadcast messages.

Source Address (Section 4)

This field indicates the sender of the message. This allows the recipient to know who sent the message (especially in case of an ACK or NAK).

Transaction ID (Section 5)

A 2-byte number. This is an "arbitrary" number that is created by the sender. The value can be zero, but it is recommended that each message have a uniquely identifiable Transaction ID.

ACK and NAK messages do not assign Transaction IDs - they simply return the ID of the original message sender.

Liquid Robotics Proprietary and Confidential

Transaction ID numbers are useful when a Payload Interface Board (PIB) sends out multiple messages. Each message should have a different Transaction ID. The response to each message will also contain the Transaction ID. This allows the sender to keep track of which messages were ACK'ed or NAK'ed.

A device may receive a message from two different sources, but with the same Transaction ID. There should not be any problem with this - the recipient should consider the address of the sender, not the Transaction ID, when responding to the message.

Except for ACK and NAK packets, consecutive valid received messages with the same Transaction ID, Source, Size, and CRC should be considered to be duplicate messages. In this case, the message recipient should simply send back duplicate responses. However, if a duplicated message requests an action, the recipient should not perform the requested action more than once.

Message Type (No ACK) (Section 6)

This is a 2-byte code that indicates the purpose and contents of this message in the least significant 15 bits (0-0x7FFF). The most significant bit (0x8000) contains a flag:

- Messages sent from Parent to Child set the flag to 1 if no acknowledgement should be sent (this is typical for broadcast or informational messages). If the flag is 0, the Child must reply with either an ACK or a NAK message.

Message Data (Section 7)

This field contains the message content to be forwarded. Some packets can contain no Message Data.

Normally, a maximum of **544** bytes is allowed, but characteristics of the recipient may impose additional length restrictions. For example, data intended to be sent out in the Iridium Short-Burst-Data format is limited to **192** bytes.

CRC-16 (Section 8)

This error-check field contains a 16-bit CRC of the message starting at the byte following the **SOF** until the byte just before the CRC field.

SOF and ESC Characters and Escaping the Message Body

Sections 2-8 of the message are escaped to prevent the SOF character (0x7E) from appearing more than once in the message. This simplifies interpretation of the incoming message since any SOF character restarts block reception.

The Escape (ESC) character is 0x7D. When either a SOF or an ESC character is used within the packet (including the CRC), it must be preceded by the ESC character and also have its bit 5 toggled. For example, for all bytes following the SOF:

0x7E → 0x7D,0x5E

0x7D → 0x7D,0x5D

The size values and size limits for the messages exclude the count of any ESC characters.

Addresses

Addresses are 2-byte values used to route messages between the CCU, Payload Interface Boards (PIBs) and other devices. These addresses are sometimes used within the body of a message, particularly when responding to broadcast messages.

Liquid Robotics Proprietary and Confidential

Address Format

Source and Destination addresses are composed of a two-byte value. The MSB is the “Board ID” value. The LSB is the “Task ID”. These are both described in more detail below.

Note that, since the address is stored in little-endian format, the actual order of storage for the address will be:

LSB	MSB
Task ID	**Board ID**

Board ID

The MSB of the address contains the “Board ID”, which has defined values as follows:

Board ID	Description
0x00	Reserved ID of the Float CCU controller
0xFF	Broadcasts the message to all boards; Some scenarios do not support broadcasts
Other	An 8-bit ID that was configured to uniquely identify a device in this vehicle

Payload Interface Boards (PIBs) contain a Board ID that is embedded in Non-Volatile Memory on the board. Custom payload boards can assign their own Task IDs. However, the Board ID must not be duplicated by any device on any single Wave Glider vehicle.

Task ID

The LSB of the address contains the Task ID, which is defined by the individual device. Payload Interface Boards (PIBs) and other non-CCU devices can select their own addresses.

The following Task IDs are pre-defined for the CCU (board ID 0):

Task ID	Description	Notes
0x00	Main process running on the CCU	
0x01	The command interface on the Float CCU	[1]
0x02	Iridium modem (send Level 2 message)	[2]
0x03	Debug Console interface	[3]
0x04	XBee interface (send Level 2 message) **(SV2 Only)**	
0x05	File transfer to shore.	[4]
0x06	Default Host (send Level 2 message)	
0x20	Forward full message to target	[5]

Notes:

[1] Level 2 messages to be executed by the CCU should be directed to Task ID **0x01**.

[2] Telemetry messages should be sent without the Level 2 transaction ID field to Task ID **0x02**. The maximum Level 2 message size is 300 bytes.

[3] Using Task ID **0x03**, a device can request to send literal text to the Console.

Liquid Robotics Proprietary and Confidential

[4] New File Available messages should be sent to Task ID **0x05**.

[5] ACKs to Write Console commands should be sent to Task ID **0x20**, prefixed by message routing instructions. See the note on Send/Forward ICD Message.

Other Considerations

Message Routing

Devices that implement this protocol can act as message routers for connected child devices. Responsibilities of the message router are:

- Check messages received and handle any messages bound for the router device, or else forward to children. If the message has an unknown destination, queue for sending to parent when polled.
- Keep track of child devices that have indicated they have messages to send. When polled for pending messages, poll those children that have messages waiting. Message routers should not buffer messages from children.
- Messages with a broadcast Board ID (i.e., **0xFF**) should be forwarded to all children in addition to being processed by the message router.
- Route the **Power Status and Control** message promptly. Broadcast notifications to children when power removal is imminent.

Timeouts and Message Usage

Timeout and retries used for the exchange of messages should be controlled with configuration variables in the device software. The settings and suggested values are:

Setting	Description	Range	Suggest
Data Link Response Timeout	The maximum time (in milliseconds) to wait for Data Link ACK or NAK before attempting retry or discarding. This time is not end-to-end but between communicating peers.	10 to 60000 ms	5000 ms
Data Link Retries	The maximum number of retries to make before failing the transaction. A parent has the option to retry. A child will not retry.	0 to100	3
Message Receive Timeout	The maximum time (in milliseconds) to wait between bytes of a message before abandoning it.	10 to 60000 ms	100 ms

Note

Messages indicated in any examples represent intended usage and were developed in response to Customer requests. While other use scenarios are possible, please consult with Liquid Robotics on the best course of action, if your requirements are not addressed.

Payload Message Types

Message IDs below **0x1000** are reserved for Liquid Robotics use. Any customer-developed messages specific to their devices must use an ID value greater than **0x1000**.

Liquid Robotics Proprietary and Confidential

The reserved Message Types are listed in the following table:

Message Type	Description	Direction	Detail	Broadcast	Reference
0x0000	NAK	Child → Parent	Unable to handle message or request	No	"Generic Payload NAK Data Packet (0x0000)" (page 157)
0x0001	ACK	Child → Parent	Successfully handled message/ request	No	"Generic Payload ACK Data Packet (0x0001)" (page 158)
0x0010	Request ID/ Enumerate	Parent → Child	Parent requests device ID and that of any attached devices	Yes	"Request ID/Enumerate (0x0010)" (page 159)
0x0015	Telemetry	Parent → Child	Forwards telemetry message to payloads that have requested it	Yes	"CCU Telemetry (0x0015)" (page 161)
0x0022	Request Status	Parent → Child	Parent requests status of Child	No	"Request Status (0x0022)" (page 161)
0x0023	*Reserved for future use*				
0x0024	Send/Forward Message	Parent → Child	Parent copies a requested message to child node	Yes	"Send/Forward Message (0x0024)" (page 163)
0x0030	Power Status and Control	Parent → Child	Parent notifies Child that power is about to turn off	Yes	"Power Status and Control (0x0030)" (page 164)
0x0040	Request Queued Message	Parent → Child	Parent requests report of any queued message	No	"Request Queued Message (0x0040)" (page 165)
0x0041	ACK/NAK Queued Message	Parent → Child	Parent forwards result of message execution	No	"ACK/NAK Queued Message (0x0041)" (page 166)

Notes:

[1] For response messages from child to parent, the high bit (bit 15) of the Original source Msg Type number can be set to indicate that there are messages queued for sending and that they should be read out using Msg Type **0x0040** (Request Queued Message).

[2] For messages from Parent to Child, the high bit of **Msg Type** (0x8000) can be set to indicate that no ACK should be sent in response to the message.

[3] If the ACK is in response to a Request Queued Message (0x0040), it can broadcast.

Payload Message Packet Descriptions

Payload Header Abbreviation

Payload Header details:

Liquid Robotics Proprietary and Confidential

Description	Bytes	Explanation
Header	9	Start-of-Frame + Header data

Start-of-Frame and Header Data details:

Field	Bytes
Start-of-Frame	1
Message Size	2
Destination Address	2
Source Address	2
Transaction ID	2

Generic Payload NAK Data Packet (0x0000)

If a recipient cannot properly handle a message or a request, it sends a NAK data packet back to the original sender.

The **Destination Address** is set to the original source address. The **Source Address** is set to the recipient's address. The **Transaction ID** value is set to the Transaction ID in the original request.

Note that the **Msg Type** of the original source message is included in the Message Data. Bit 15 (**0x8000**) can be set to indicate that there are additional NAK response messages.

Depending on the type of NAK, up to **541** bytes of additional information can be returned. Some responses can return an entire Level 2 NAK message including the CRC.

Description	Bytes	Explanation
Header	1	Start-of-Frame
	2	Message Size
	2	Destination Address
	2	Source Address
	2	Transaction ID
Msg Type	2	0x0000
Message Data	2	Original source Msg Type number (bit 15 set if additional messages available)
	1	Message Code: **0**: Returning Level 2 NAK Message **1** → **3**: Reserved For Liquid Robotics **4**: Program Download Requested **5** → **31**: Reserved For Liquid Robotics **32** → **255**: Customer definable
	≤ 541	NAK details or reply data (if any).
Error Chk	2	CRC-16

NAK Message Types are as follows:

Returning NAK Data Message

If the NAK Message Code is 0, the response contains a Level 2 enumerated NAK message including the CRC.

Program Download Requested (Not Implemented)

If the NAK Message Type = 4, the following data structure is returned in the message field:

Description	Bytes	Detail
Program Download Reason	1	**0x00**: Responding to request to load newly selected program **0x01**: Responding to request to load default program **0x02**: CRC did not match **0x03**: MD5 did not match **0x04**: Bad configuration in EEPROM **0x05**: Tentative image execution failed
CRC Expected	2	CRC expected for program image
CRC Actual	2	CRC actually computed
MD5 Expected	16	MD5 signature expected for program image
MD5 Actual	16	MD5 signature actually computed (Can be zero if boot loader is not able to compute MD5)
Number of Failed Starts This Image	2	Number of times the boot loader has attempted to start the program, but determined that it had failed (Never increments above 0xFFFF)
Program Type Loaded	1	**0x00**: Non-default program loaded **0x01**: Default program loaded

Generic Payload ACK Data Packet (0x0001)

If a message is successfully handled, an ACK data packet will be sent to the original message sender.

The **Destination Address** is set to the original source address. The **Source Address** is set to the recipient's address. The **Transaction ID** value is set to the Transaction ID in the original request.

All ACK responses return the **Msg Type** of the original source message at the beginning of the Message Data. Bit 15 (**0x8000**) can be set to indicate that there are additional ACK response messages.

Up to **542** bytes of additional information can be added. Some ACK responses can return an entire Level 2 message, including the CRC.

Description	Bytes	Explanation
Header	1	Start-of-Frame
	2	Message Size
	2	Destination Address
	2	Source Address

Liquid Robotics Proprietary and Confidential

Description	Bytes	Explanation
	2	Transaction ID
Msg Type	2	0x0001
Message Data	2	Original source Msg Type number (bit 15 set if additional messages available)
	2	Command Format (0x0001 defined in Appendix A: *Command Descriptions*)
	≤ 542	ACK details or reply data (if any).
Error Chk	2	CRC-16

Request ID/Enumerate (0x0010)

This payload message requests identifying information about a Device. The message contains bit-maps that indicate which protocols and commands are supported by the CCU. Each Device responds with an ACK or NAK.

An ACK response contains information specifying which CCU protocols the Device supports.

A NAK response indicates that the Device doesn't support any of the protocols. If there is a NAK (or no response), the CCU will not send any messages to the Device.

For this message, the **Message Data** contains two fields:

A 2-byte bit-mask specifies what communication protocols are supported by the CCU.

A 2-byte bit-mask specifies what command formats are supported by the CCU.

The Request ID soliciting command format is:

Description	Bytes	Explanation
Header	9	Start-of-Frame + Header data
Msg Type	2	0x0010
Message Data	2	Bit mask of protocols supported by the Float CCU. 0x0001: Existing Device Protocol. All other bit mask values are reserved by Liquid Robotics.
	2	Mask of command formats accepted by the Float CCU. 0x0001 = Existing Vehicle-to-remote applications commands.
Error Chk	2	CRC-16

In the specific case where the message is sent as a broadcast, the device should resend the request to its children and should concatenate responses. The receiving devices should only respond if they can communicate using at least one of the Float CCU protocols.

ACK Reply to Enumerate Message

Multiple device information blocks can be returned when multiple responses are aggregated, for example, in the case of a Payload Interface Board (PIB) that has daisy-chained devices or other PIBs. Up to 5 devices are currently supported on each payload port. This is done using an ACK with the format shown below:

Liquid Robotics Proprietary and Confidential

Description	Bytes	Data Type
Header	9	Start-of-Frame + Header data
Msg Type	2	0x0001
Message Data	2	Original source Msg Type number (bit 15 set if additional messages available)
	2	Number of devices responding (maximum 5)
	2	Number of devices in this message (must be equal to the number of devices responding)
	50	1st Device Information Block (see below)
	50	2nd Device Information Block
	. . .	. . . .
	50	Nth Device Information Block (if necessary)
Error Chk	2	CRC-16

Device Information Block Format

All Device Information Blocks use the following format:

Description	Bytes	Value	Detail
Version	2		Device Information Version Format number: **1:** This format
Device Type	2		**0x0000** – **0x1000** are reserved for Liquid Robotics **0x1001** – **0x7FFF** are allowed for customer use
Address	2		[Board ID + Task ID] Address of responder [1]
Serial Number	6	ASCII	Optional Manufacturer's serial number for the device, as ASCII characters. Padded on right with zeros (null terminators). A completely null string is allowed.
Location/Port	1	0	This field is reserved and should be set to zero (0)
Polling Frequency	4	Any	The requested number of seconds between polling operations, using the Request Status message
Want Extra Info	1	0x00 – 0x0F	Bit masks indicating whether a component wants to receive event notifications from the Float. **0x01**: Telemetry information desired **0x02**: Power information desired **0x04**: Event information desired **0x08**: Command ACK/NAK desired The upper 4 bits (0xF0) are reserved for Liquid Robotics and should be set to 0
Firmware (Major)	1		Integer
Firmware (Minor)	1		Integer
Firmware (Revision)	2		Integer

Liquid Robotics Proprietary and Confidential

Description	Bytes	Value	Detail
Description	20	ASCII	Human-readable description of device. ASCII text field, padded on the right with spaces

CCU Telemetry (0x0015)

If requested in a Device Information Block, the Float CCU sends Telemetry to each device at a user-configurable rate. The rate is set as the fastest of all the rates (shortest interval) requested in the Device Information Blocks.

The Telemetry message sent to the devices is a duplicate of the Level 1 telemetry sent to shore using a relay link. The structure ID from the telemetry message determines how the message content is decoded. The message packet format is:

Description	Bytes	Data Type
Header	9	Start-of-Frame + Header data
Msg Type	2	0x0015
Message Data	1	Structure ID for Telemetry data
	≤ 544	Level 1 Telemetry data
Error Chk	2	CRC-16

The response to this message is either an ACK or NAK with message content empty. As with any ACK/NAK, a device can signal if there are messages queued to be read. Sometimes telemetry is forwarded using a broadcast and the “no ack” flag is set, in which case no reply is sent.

Request Status (0x0022)

This is the basic polling message. It is also used to update various commonly needed values.

Description	Bytes	Data Type	Detail
Header	9	Start-of-Frame + Header data	
Msg Type	2	0x0022	
Message Data	6	Time (UTC)	One byte per field. Bytes in order: 0: Second (0 –59) 1: Minute (0 – 59) 2: Hour (0 – 23) 3: Day of Month (1 – 31) 4: Month (1 – 12) 5: Year since 2000 (0 – 255)
	4	Latitude	-90° < Latitude ≤ +90° Current Latitude encoded as a 32-bit floating-point number. Note [1]

Liquid Robotics Proprietary and Confidential

	4	Longitude	-180° < Longitude ≤ +180° Current Longitude encoded as a 32-bit floating-point number. Note [2]
	1	Fix Valid?	Single character: 'Y' = Latitude and Longitude are valid 'N' = Latitude and Longitude are not valid
Error Chk	2	CRC-16	

Notes:

[1] Latitude = *degrees**600000 + *minutes**10000 + *minute_fraction*. [-90 < *degrees* ≤ 90] [0 ≤ *minutes* < 60] [0 ≤ *minute_fraction* < 10000] If *degrees* < 0, location is in the Southern hemisphere

[2] Longitude = *degrees**600000 + *minutes**10000 + *minute_fraction*. -180 < *degrees* ≤ 180. 0 ≤ *minutes* < 60. 0 ≤ *minute_fraction* < 10000. If *degrees* < 0, location is in the Western hemisphere

ACK Reply to Request Status

The response to the Request Status is returned in an ACK packet.

As with any ACK, the device can signal if there are messages queued to be read.

In the case of a broadcast, the CCU or a Payload Interface Board (PIB) will retransmit the message to its connected devices and will accumulate responses. Depending upon the message, multiple response messages may be needed.

The Status ACK returns one or more blocks of status information for the responding devices. As described below, there are two Status Block versions with different formats.

Description	Bytes	Data Type
Header	9	Start-of-Frame + Header data
Msg Type	2	0x0001
Message Data	2	Original source Msg Type number (bit 15 set if additional messages available)
	2	Total number of responses (must be 1)
	2	Number of responses in this message (must be 1)
	12	1st Status Block
CRC-16	2	

Status Block, Version = 1

The only defined return type is version 1. The Version and Address fields must be set with real values, but the remainder of the block can be set to zero (0) if there is nothing to report.

Description	Bytes	Explanation
Version	2	0x0001
Address	2	Address of responding device
Alarms	2	Alarm flags (16 bit-field flags)
Leak Sensor 1	1	Status of leak sensor #1

Liquid Robotics Proprietary and Confidential

Description	Bytes	Explanation
Leak Sensor 2	1	Status of leak sensor #2
Humid Temp **(SV2 Only)**	1	Temperature from humidity sensor (°C)
Relative Humidity **(SV2 Only)**	1	% Humidity
Pressure Temp	1	Temperature from pressure sensor (°C)
Pressure	1	Air pressure (kPa)

Send/Forward Message (0x0024)

This message is used to send or to forward Level 2 data packets to a device (messages that conform to the vehicle-to-remote applications protocol). These packets have either been requested as copies (of telemetry, for example) or are meant for interaction with systems external to the vehicle. For example, the Level 2 Write Console Script message is sent using Send/Forward Message.

The format is:

Description	Bytes	Data Type	Detail
Header	9	Start-of-Frame + Header data	
Msg Type	2	0x0024	
Message Data	1	Flag	Indicates whether this is a Forward or a Send as: **0x00**: Forward [1] **0x01**: Send [2]
	2	Command Format	Mask of command formats accepted that this message conforms to: **0x0001**: vehicle-to-remote applications commands
	≤ 541	Level 2 Packet	The Level 2 packet containing the message, including the CRC.
Error Chk	2	CRC-16	

Notes:

[1] In the "Forward" case the command is only informational and typically no ACK is requested.

[2] In the "Send" case, the command is expected to be executed.

ACK for Forward

If an ACK is requested for the “Forward” case (not typical), the response is an empty ACK message.

ACK for Send

If an ACK is requested for the “Send” case, the ACK packet carries the response type expected by the Level 2 message encapsulated as follows:

Description	Bytes	Data Type
Header	9	Start-of-Frame + Header data
Msg Type	2	0x0001

Liquid Robotics Proprietary and Confidential

Description	Bytes	Data Type
Message Data	2	Original source Msg Type number (bit 15 set if additional messages available)
	≤ 542	Level 2 response message, including the CRC
Error Chk	2	CRC-16

Note that responding to Write Console is a special case: respond with a basic ACK (Level 2 type 0x13) and the transaction ID, but with a zero-length message payload. Responses to the Write Console message are queued up as follows:

Description	Bytes	Data Type
Header	9	Start-of-Frame + Header data
Msg Type	2	0x0001
Message Data	2	Request Queued Message type (0x40) (bit 15 set if additional messages available)
	1	Message format (**2**; 1= remote applications-to-vehicle message, 2=vehicle-to-remote applications message)
	2	Destination address (saved from the source of the Send/ Forward Message)
	2	Source address
	≤ 537	Level 2 response message, including the CRC
Error Chk	2	CRC-16

Power Status and Control (0x0030)

This message is sent to notify a device of an intended power shutdown.

Description	Bytes	Data Type	Detail
Header	9	Start-of-Frame + Header data	
Msg Type	2	0x0030	
Message Data	1	Command Type	Numerical flag value: **0x00**: Notification of power Off
	1	Board Affected	Reserved value: **0x00**: Recipient Device
	1	New Power State	**0x00**: Power Off
Error Chk	2	CRC-16	

ACK Response to Power Status and Control

Multiple response blocks can be returned in an ACK with the format shown below.

Description	Bytes	Data Type
Header	9	Start-of-Frame + Header data
Msg Type	2	0x0001

Description	Bytes	Data Type
Message Data	2	Original source Msg Type number (bit 15 set if additional messages available)
	2	Number of total devices responding (1)
	2	Number of responses in this message (1)
	6	Power State Record
Error Chk	2	CRC-16

Power State Record Format

The Power State Records all use the following format:

Description	Bytes	Detail
Address	2	Address of responder
Board ID Number	1	The ID of the board number affected
Power State	1	Flag value: **0x00**: Board is shut down **0x01**: Board is currently operating
Power Off Delay	1	The minimum number of seconds to wait after receiving the ACK before powering Off
Power On Delay	1	The minimum number of seconds to wait after powering Off before powering On again

Request Queued Message (0x0040)

This packet is sent when a previous message has indicated that there is at least one message waiting to be read.

Description	Bytes	Explanation
Header	9	Start-of-Frame + Header data
Msg Type	2	0x0040
Error Chk	2	CRC-16

NAK Response to Request Queued Message

Normally, there is no need to negative-acknowledge Request Queued Message since it is only sent when the board has previously indicated that there is a message waiting. If there is no data to be read, the following NAK is returned:

Description	Bytes	Explanation
Header	9	Start-of-Frame + Header data
Msg Type	2	0x0000
Message Data	2	Original source Msg Type number
	1	0x02 (Invalid Data)
	2	Length = 0x00

Liquid Robotics Proprietary and Confidential

Description	Bytes	Explanation
Error Chk	2	CRC-16

ACK Response to Request Queued Message

If there is a message waiting, the response format is:

Description	Bytes	Explanation
Header	9	Start-of-Frame + Header data
Msg Type	2	0x0001
Message Data	2	Original source Msg Type number (bit 15 set if additional messages available)
	≤ 542	Data message to be passed to destination
Error Chk	2	CRC-16

The format of the data message is dependent on the target address.

ACK/NAK Queued Message (0x0041)

When a payload executes a command by sending the appropriate message to a target device, this message carries back the result of that command. For example, a payload could:

1. Queue up a Set Rudder Position command, targeting address 1 (the Float CCU command interpreter). The format to target the command interpreter (address 1) is:

Description	Bytes	Explanation
Header	9	Start-of-Frame + Header data
Msg Type	2	0x0001
Message Data	2	Original source Msg Type number (bit 15 set if additional messages available)
	2	Message format (1)
	≤ 540	Level 2 data to be executed
Error Chk	2	CRC-16

2. Respond to Request Status and indicate that there are messages waiting to be read.
3. Respond to Request Queued Message with the queued Set Rudder Position message.
4. When the command is executed by the Float CCU, it sends ACK/NAK Queued Message with the result of the command execution.

Description	Bytes	Explanation
Header	9	Start-of-Frame + Header data
Msg Type	2	0x0001
Message Data	2	Command Format: **0x0001:** ICD Format
	2	Response Code:

Liquid Robotics Proprietary and Confidential

Description	Bytes	Explanation
		0x00: ACK **0x01**: NAK - Unknown Command **0x02**: NAK - Invalid Data **0x03**: NAK - Unable to Complete **0x04**: NAK - Unknown Format
	≤ 540	The response message (if any)
Error Chk	2	CRC-16

The response to this message is an ACK with no payload, or a NAK with no accompanying data.

Message Protocol Use Cases

Different use scenarios and the sequence of actions taken for each are shown in the following sections.

Bold text indicates a command (or ACK/NAK response)

Italic text indicates an action taken

[Italic] indicates a comment

Bold text indicates a command (or ACK/NAK response)

Italic text indicates an action taken

[Italic] indicates a comment

Normal Polling Update Time and Location

This case shows a Payload Interface Board (PIB) trying to send a Request Status command to a Device.

PIB		Device
Request Status	→	*Data link verifies CRC and transport packet.*
		[Device interprets message, sets timestamp and saves location.]
Data link verifies CRC and transport packet.	←	**ACK** (*single response, no queued messages, version 0*)
[PIB handles status request reply.]		

Start-up Identification Broadcast

PIB		Device
[PIB waits for a specified or assumed time after start-up before attempting to communicate.]		
Request ID/Enumerate	→	*Data link verifies CRC and transport packet.*
		[Device interprets message and verifies compatibility with Float CCU Protocol

Liquid Robotics Proprietary and Confidential

PIB		Device
		Mask. Device reads its Serial Number and Board ID from non-volatile RAM or other storage.]
Data link verifies CRC and transport packet.	⟵	**ACK** (*single response, no queued messages*)
[PIB forwards response to requester, then maps Address field of response to a Port to build a routing table for future messages.]		

Device Receives File from Remote Application

PIB		Device
[remote application sends a file tagged for this Board ID. The file is stored in the file system on the Float CCU. After the write completes, the file is checked, and the process of sending the file to the device begins. Both the PIB and the Device must be powered on for the file to be forwarded.]		
Send/Forward Message *(send Initialize File Write)*	⟶	*Message layer verifies the message CRC and transport fields.*
		[Space is reserved for the file on the device, and initial verification checks are made. Application-specific logic can be applied to determine what the file contains based on the file name or the content of the first block of data.]
Message layer verifies the message CRC and transport fields.	⟵	**ACK** *(Write File ACK)*
[ACK message is forwarded to Float CCU, which schedules the next packet.]		
Send/Forward Message *(send Write File Block)*	⟶	*Message layer verifies the message CRC and transport fields.*
		[Data is written to file or storage.]
Message layer verifies the message CRC and transport fields.	⟵	**ACK** *(Write File ACK)*
[ACK message is forwarded to Float CCU, which schedules the next packet.]		
[Float CCU continues to send Write File Block until all blocks are transferred.]		
Send/Forward Message *(send Get File List, indicating the specific file that was sent and asking for the file to be checked)*	⟶	*Message layer verifies the message CRC and transport fields.*
		[The file CRC and MD5 are checked against the expected values from Initialize File Write.]
Message layer verifies the message CRC and transport fields	⟵	**ACK** *(Get File List Data)*

Liquid Robotics Proprietary and Confidential

PIB		Device
[Float CCU sends File At Destination event to remote applications.]		*[File is now resident on device; this event can be used to interpret the file if it is actually a command or a parameter set.]*

Send File from Device to Remote Application

PIB		Device
		[Device finishes creating a new data set to be sent to remote applications.
Request Status	→	*Message layer verifies CRC and transport packet.*
		[Interprets message, sets timestamp and saves location.]
Message layer verifies CRC and transport packet.	←	**ACK** *(single response, have queued messages, version 0)*
[Handles status request reply.]		
Request Queued Message	→	*Message layer verifies CRC and transport packet.*
		[Selects the next message in its queue to be sent.]
Message layer verifies CRC and transport packet.	←	**ACK** *(response with New File Available message tagged for File Transfer to remote application Task ID)*
[Float CCU sends New File Available message to shore side using the configured method.]		
ACK/NAK Queued Message *(simple ACK)*	→	*Message layer verifies CRC and transport packet.*
Message layer verifies CRC and transport packet.	←	**ACK** *(no accompanying data)*
Float CCU is informed of result.		
[Remote application evaluates whether the file should be uploaded according to rules configuration, and initiates a Read File request. Float CCU starts a Read File operation to buffer the file in the Float CCU file system, before transmitting.]		
Send/Forward Message *(send Read File, indicating the specific file that was advertised)*	→	*Message layer verifies CRC and transport packet.*
		[Device allocates a read operation context that can return chunks of the file. Saves the requester and the transaction ID as all resulting blocks will be returned using the same path.]
Message layer verifies CRC and transport packet.	←	**ACK** *(Read File ACK containing the first block of data requested. Indicates more data available, if more blocks are to be read from the file.)*

Liquid Robotics Proprietary and Confidential

PIB		Device
[Float CCU stores the file chunk in its file system and schedules next check for this device.]		
Request Queued Message	→	*Message layer verifies CRC and transport packet.*
		[Device selects the next message in its queue to be sent, which is a Read File ACK.]
Message layer verifies CRC and transport packet.	←	**ACK** *(responds with Read File ACK message tagged with requester's address)*
[Float CCU stores the file chunk in its file system and schedules next check for this device.]		
[The Float CCU continues its scheduled polling and receives all chunks of the file. It then begins to respond to the Read File request and sends the file in chunks to the remote requester. When the requester has received the entire file, it issues Read File Transaction Closed to the Float CCU, which deletes the copy of the device's file from its file system.]		

Device Receives Power Off Notification

PIB		Device
Power Status and Control *(Notification of Power Off)*	→	*Message layer verifies CRC and transport packet.*
Message layer verifies CRC and transport packet.	←	**ACK** (*single response, power is ON, power off delay 3 seconds, power on delay 60 seconds*)
[Float CCU schedules power off.]		*[Prepares to shut down.]*
[Float CCU turns power off.]		

Send Message to Console

PIB		Device
Request Status	→	*Message layer verifies CRC and transport packet.*
		[Device interprets message, sets timestamp and saves location.]
Message layer verifies CRC and transport packet.	←	**ACK** *(single response, have queued messages, version 0)*
[PIB handles status request reply.]		
Request Queued Message	→	*Message layer verifies CRC and transport packet.*

Liquid Robotics Proprietary and Confidential

PIB		Device
		Device selects the next message in its queue to be sent.
Message layer verifies CRC and transport packet.	←	**ACK** *(responds with Write Console to Client message tagged for the command interface task ID on the Float CCU)*
[Float CCU sends message to console.]		
[Send/Forward Message for Write Console Input to the vehicle is likely to be sent without request for ACK, in which case the scenario stops here.]		
ACK/NAK Queued Message *(passing result of Initialize File Write request) (marked as No ACK)*	→	*Message layer verifies CRC and transport packet.*

Chapter 9. Payload Development Mechanical and Electrical Considerations

This chapter provides information on:

- Mechanical and Electrical parameters that may impact the feasibility of a payload integration project.
- Optional Wave Glider accessories designed to save time with payload integration projects.

Float Payload Considerations

There are several parameters that determine float payload integration feasibility:

- Weight
- Center-of-Gravity
- Sail Area Considerations
- Deck Mounting Considerations
- Towing Considerations

Weight

The maximum allowed weight of the Float with standard accessories and all float-mounted payloads is 113.4 Kg (250 lb).

Table 9.1. Weight of Float and Standard Accessories

Component	Weight
Float with Skeg, Handles, Cover Plate, Mast Plate, Motion Isolator, Lift Points, Resilient Tracker, Speed Sensor, Octopus Cables, Cable Guides and Foam Inserts.	35.6 Kg (78.5 lb)
Three Solar Panels (with power cables)	11.6 Kg (25.5 lb)
CCU (includes AIS)	17.0 Kg (37.5 lb)
Weather Station and Marker Light on 1 meter Mast with Flag	1.37 Kg (3 lb)
Radar Target Enhancer Mast assembly	0.91 Kg (2 lb)
Total Weight of Float and Liquid Robotics Standard Accessories	66.45 Kg (146.5 lb)

Assuming a total weight of 66.45 Kg (146.5 lb) for the Float with Liquid Robotics standard accessories, the weight of payloads must not exceed 46.95 Kg (103.5 lb).

Table 9.2. Maximum Payload Weight

Component	Weight
Maximum combined weight for all payloads added to the Wave Glider SV3 Float (including Liquid Robotics standard Float accessories).	46.95 Kg (103.5 lb)

Liquid Robotics Proprietary and Confidential

Center-of-Gravity

For maximum efficiency, the horizontal center-of-gravity of the Float must be within a narrow range.

The center-of-gravity value can be verified by placing the Float with the Umbilical and Sub attached in calm water and checking that the Mast Plate pitch angle does not exceed the fore and aft ranges shown in Figure 9.1 and Figure 9.2 below. This procedure is more accurate if performed in salt water, but it can be done in fresh water. Ideally, the Float should be slightly nose-high.

Figure 9.1. Float Max Forward Center-of-Gravity

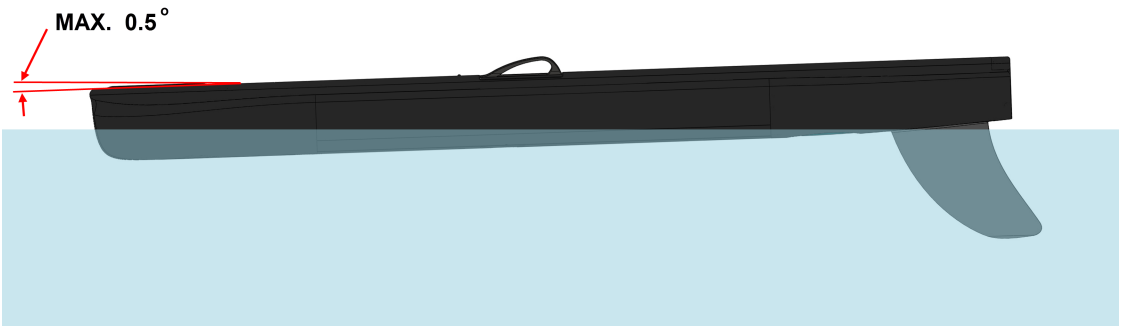
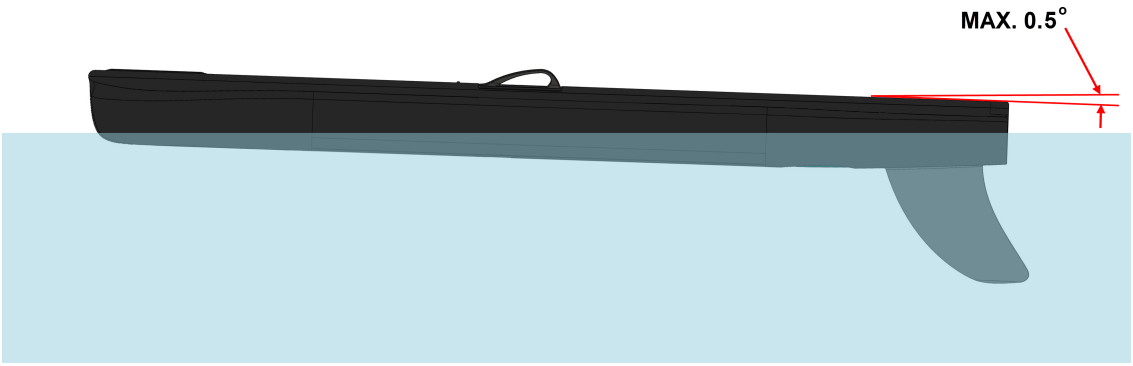


Figure 9.2. Float Max Aft Center-of-Gravity



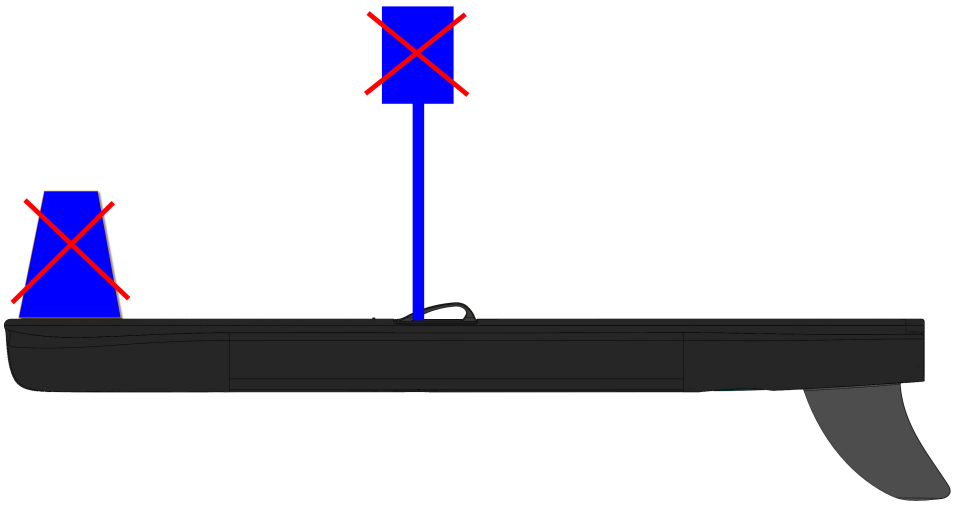
IMPORTANT - Payloads must be ballasted, if necessary, so the Float is **+/- 0.2** degrees from port side to starboard side, when resting in calm water (it cannot list).

Sail Area Considerations

There are limits to the amount of sail area that can be mounted on top of the Float. Payloads with large sail areas should be avoided. Although the underwater Sub acts like a large keel to stabilize the Float, tall masts with large or heavy tops should also be avoided. Contact Liquid Robotics Customer Support before mounting objects on the Float that could create significant sail effects or impact the stability of the Float.

Liquid Robotics Proprietary and Confidential

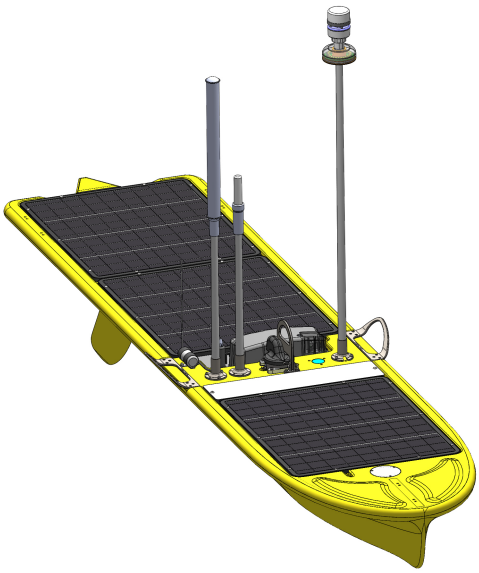
Figure 9.3. Possible Sail Area Problems



Deck Mounting Considerations

Mounting payloads to the top of the Float deck may be acceptable, if the payload does not significantly shadow the solar panels (for more information on see "Shading" (page 11). Liquid Robotics recommends locating Float-mounted payloads on the Mast Deck, as shown in Figure 9.4. Consult with the Liquid Robotics Mechanical Engineering Organization before mounting payloads in any other location on the Float deck.

Figure 9.4. Deck Mounted Payloads

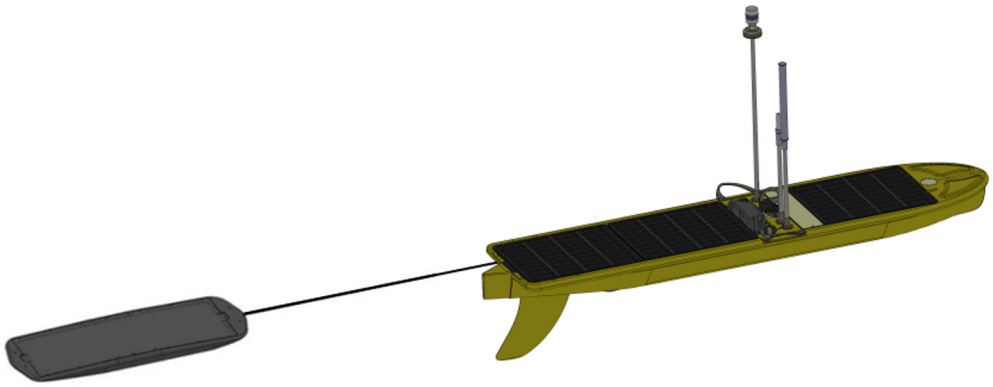


Liquid Robotics Proprietary and Confidential

Towing Considerations

Small payloads can be towed behind the Float, although the additional drag will slow the vehicle. Consult with the Liquid Robotics Mechanical Engineering Organization before attempting to tow a payload.

Figure 9.5. Towing a Payload Behind Float



Payload Power Considerations

The Wave Glider has specific limitations on the power used by payloads.

Power Requirements - Evaluating Payload Integration Feasibility

To determine the feasibility of a Wave Glider payload integration project, you need to evaluate the payload power requirements relative to the power that is available.

Payload Maximum Currents

Each CCU General Purpose Expansion Port connector is regulated to 14-25 volts at 10 amps.

Each CCU Sensor Port connector is regulated to 13.4 volts at 3 amps.

The CCU High Voltage Port is regulated to 48 volts at 2 amps.

The CCU Umbilical connector provides a power source to the Thrudder Module that is regulated to a maximum of 1 amp.

The Thrudder Module Payload Expansion Port connector is regulated to a maximum of 1 amp.

Attempting to exceed the current limits on any CCU or Thrudder connector will cause a shut-down of the voltage to that output.

Liquid Robotics Proprietary and Confidential

Power Requirements - Evaluating Wave Glider Mission Feasibility

To evaluate the feasibility of a payload integration project relative to a specific mission, you need to determine:

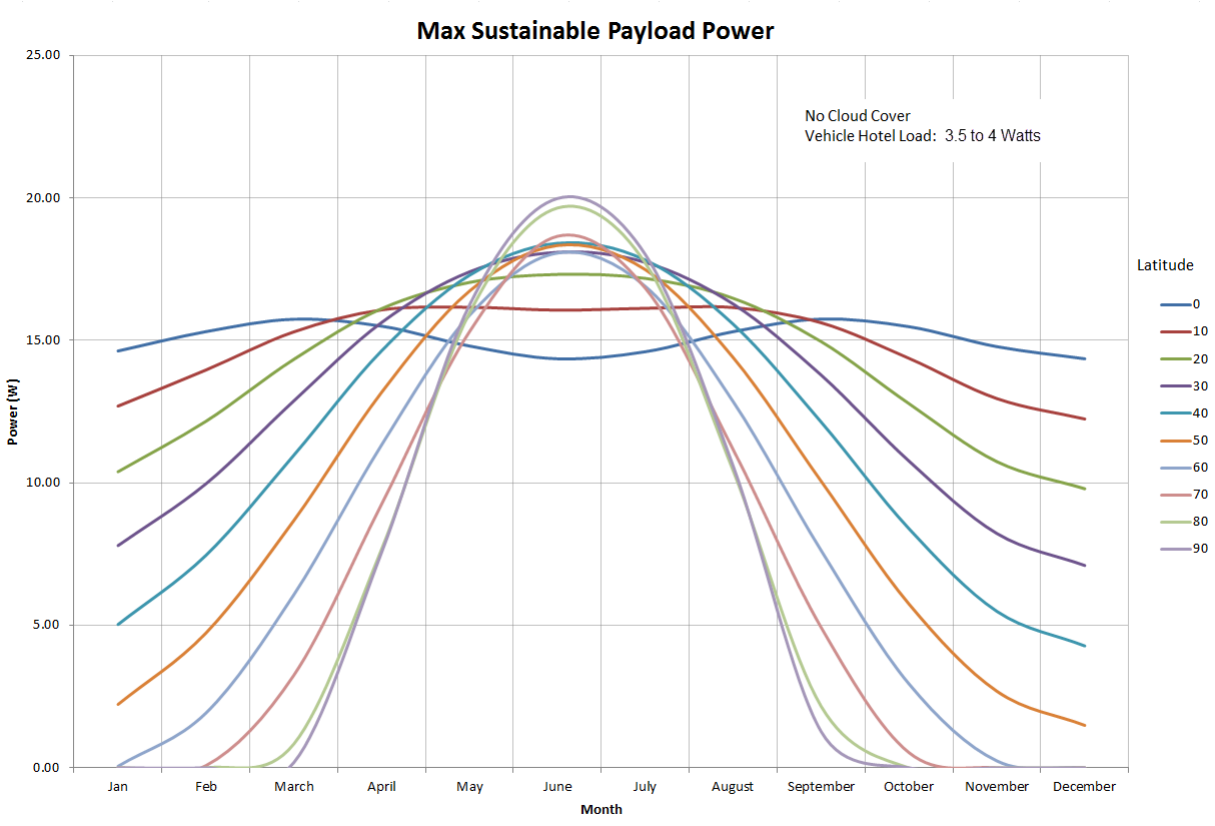
- The average and maximum power draw for the Wave Glider system including all payloads.
- The average solar power harvesting capability of the Wave Glider solar panels for the locale and time of year the mission will be active.

Sustainable Payload Power

The Wave Glider™ derives all of its long-term operating power from the solar panels. Under ideal conditions, the three solar panels on the Wave Glider can produce 156 Watts of power (see "Solar Panels" (page 10) for solar panel specifications). Depending on locale and time of day, the solar panels will harvest less energy.

Figure 9.6 is a conceptual drawing that shows the computed power available to a Wave Glider at different northern latitudes and dates. The graph assumes that cloud cover reduces the solar flux by 25%. However, it does not take into account the larger atmospheric absorption at far-north latitudes.

Figure 9.6. Solar Power Relative To Time and Location



For sustained missions, the average energy used by the CCU and payload(s) should not exceed the average energy input from the solar panels. Allowance also has to be made for the possibility that, due

Liquid Robotics Proprietary and Confidential

to storms and cloud cover, the solar energy may be considerably less than average for several days at a time.

To the extent that the average amount of energy used by the CCU and the payload(s) exceeds the average energy input from the solar panels, the battery charge will trend downward, and will eventually be unable to supplement the solar energy input to keep the mission going.

For short duration missions, if the average amount of energy used by the CCU and the payload(s) exceeds the average energy input from the solar panels, it may be possible to complete the mission before the batteries are depleted. There is also the option to increase battery capacity by installing one or more Auxiliary Power Units (APUs). See “Auxiliary Power Unit (APU)” (page 185) for more information.

Important

This information is provided to illustrate in general how location and time of year impact solar power harvesting. There are several variables involved in more accurately predicting how long a Wave Glider mission can be sustained based on latitude, time of year, power draw, battery capacity, and other considerations. If you are concerned about mission sustainability, consider hiring Liquid Robotics to develop a sustainable mission plan.

Adapting Wave Glider SV2 Payloads for use on the Wave Glider SV3

SV3 payloads are supported on SV3 vehicles only. SV2 payloads are supported on SV2 and SV3 vehicles, however, there are mechanical and electrical considerations that must be addressed to adapt a Wave Glider SV2 payload for use on a Wave Glider SV3 vehicle.

Mechanical Considerations

SV3 payload bays are larger and shaped differently than SV2 payload bays. Liquid Robotics offers mounting kits for securely installing standard SV2 forward, aft, and short aft payload dry boxes in the payload bay of an SV3 vehicle. All three mounting kits include:

- Two ABS spacers with titanium mounting hardware attach to the mounting rails of the SV3 float providing a secure attachment point for the flange of the SV2 payload dry box.
- A custom milled rigid foam spacer fits firmly under the SV2 payload dry box.

Liquid Robotics Proprietary and Confidential

Figure 9.7. SV2 Forward Payload Dry Box - SV3 Mounting Kit (Liquid Robotics Part Number 04731)

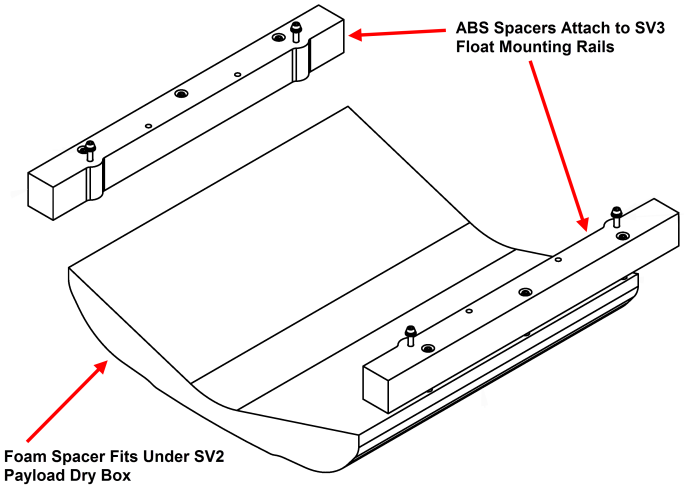
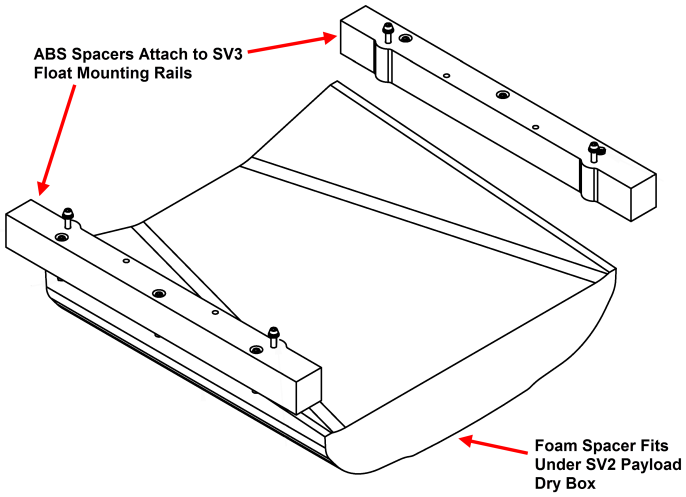
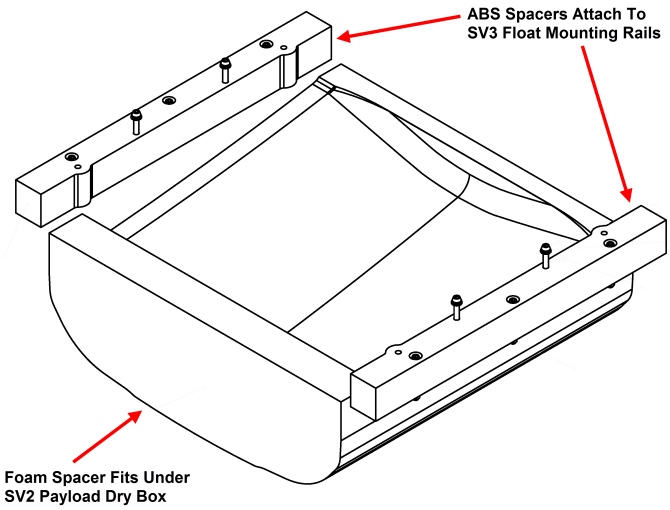


Figure 9.8. SV2 Aft Payload Dry Box - SV3 Mounting Kit (Liquid Robotics Part Number 04732)



Liquid Robotics Proprietary and Confidential

Figure 9.9. SV2 Short Aft Payload Dry Box - SV3 Mounting Kit (Liquid Robotics Part Number 05224)



Electrical Considerations

Wave Glider SV2 payloads can be adapted for use with the Wave Glider SV3.

Wave Glider SV2 payloads that run on a 13.4 volt power supply and communicate directly with the SV2 C&C can be connected to the S1, S2, and S3 Sensor ports on the SV3 CCU (the S4 connector is dedicated for use with the Weather Station). The SV3 Sensor ports support the same electrical connectors and the same payload communications protocol that is used on the SV2 Payload ports, but the Wave Glider SV3 Sensor ports are NOT identical to the SV2 Payload ports:

- There are no AUX lines on the SV3 Sensor ports, instead pins 5 and 6 in the SV3 Sensor port connectors support the multi-protocol signal capabilities that are provided by the SMC - specifically: RS-232, RS-422, and RS-485. See “S1, S2, S3, S4 - Sensor Connectors” (page 47) for more information. If you are considering adapting an SV2 payload that uses the SV2 C&C Payload connector AUX lines (pins 5 and 6), consult with the Liquid Robotics Support Organization to determine how your payload might be adapted to an SV3 electrical environment.

- The GPS baud rate on SV2 vehicles is 4800 baud. On SV3 vehicles the GPS baud rate is 9600 baud.

- The GPS report on SV2 vehicles is different than the GPS report on SV3 vehicles:

  – The order of the sentences is different.

    The sentence order for SV2 is: ZDA, GGA, VTG, GST, RMC.

    The the sentence order for SV3 is: RMC, VTG, GGA, GLL, GST, ZDA.

  – The SV2 GPS reports 5 sentences - the SV3 GPS reports 6 sentences.

  – The SV3 GPS does not report some of the fields in the GLL, GST, and VTG sentences that are included in the SV2 GPS report.

Sample SV2 GPS Report.

Liquid Robotics Proprietary and Confidential


```
$GPZDA,234041.00,25,11,2013,,*61
$GPGGA,234041.00,3724.65048,N,12200.26464,W,2,11,0.83,00000,M,-025,M,,*52
$GPVTG,000.0,T,344.4,M,000.00,N,000.00,K,D*21
$GPGST,234041.00,,127.89,127.88,071.4,011.31,011.31,011.31*48
$GPRMC,234041.00,A,3724.65048,N,12200.26464,W,000.00,000.0,251113,15.6,E,D*2B
```

Sample SV3 GPS Report.

```
$GPRMC,234821.00,A,3724.64995,N,12200.26413,W,0.026,,251113,,,D*6E
$GPVTG,,T,,M,0.026,N,0.048,K,D*2E
$GPGGA,234821.00,3724.64995,N,12200.26413,W,2,12,0.68,10.4,M,-29.9,M,,0000*52
$GPGLL,3724.64995,N,12200.26413,W,234821.00,A,D*76
$GPGST,234821.00,9.2,,,,0.63,0.46,1.3*79
$GPZDA,234821.00,25,11,2013,00,00*6F
```

The electrical specifications, including pin-outs, for SV2 C&C Payload connectors is available in the *Standard Electrical Connections* section of the *Wave Glider® SV2 Interface Control Description*.

For SV3 CCU Sensor Port electrical specifications, including pin-outs, see "S1, S2, S3, S4 - Sensor Connectors" (page 47).

Modular Payload Dry Boxes

Liquid Robotics Modular Payload Dry Boxes provide a dry environment for payload components and optionally include electronics for electrical and communications connectivity with the CCU.

The forward and aft Float payload bays can be divided into 10 equal sized spaces known as Modular Payload Units (MPUs) as illustrated in Figure 2.2. The two standard Modular Payload Dry Boxes available from Liquid Robotics conform to the cross-sectional shape of the payload bays in the Float and take up a total of either 1 MPU or 3 MPUs of space.

Liquid Robotics standard Modular Payload Dry Boxes:

- **1-MPU** - This dry box takes up 1 modular payload unit of space in a Wave Glider Float payload bay. See "1-MPU Payload Dry Box" (page 182) for more information.
- **3-MPU** - This dry box takes up 3 modular payload units of space in a Wave Glider Float payload bay. See "3-MPU Payload Dry Box" (page 184) for more information.

Note

Liquid Robotics payload integration partners can request solid models of the 1-MPU and 3-MPU to utilize in their payload designs.

Each Liquid Robotics Modular Payload Dry Box includes:

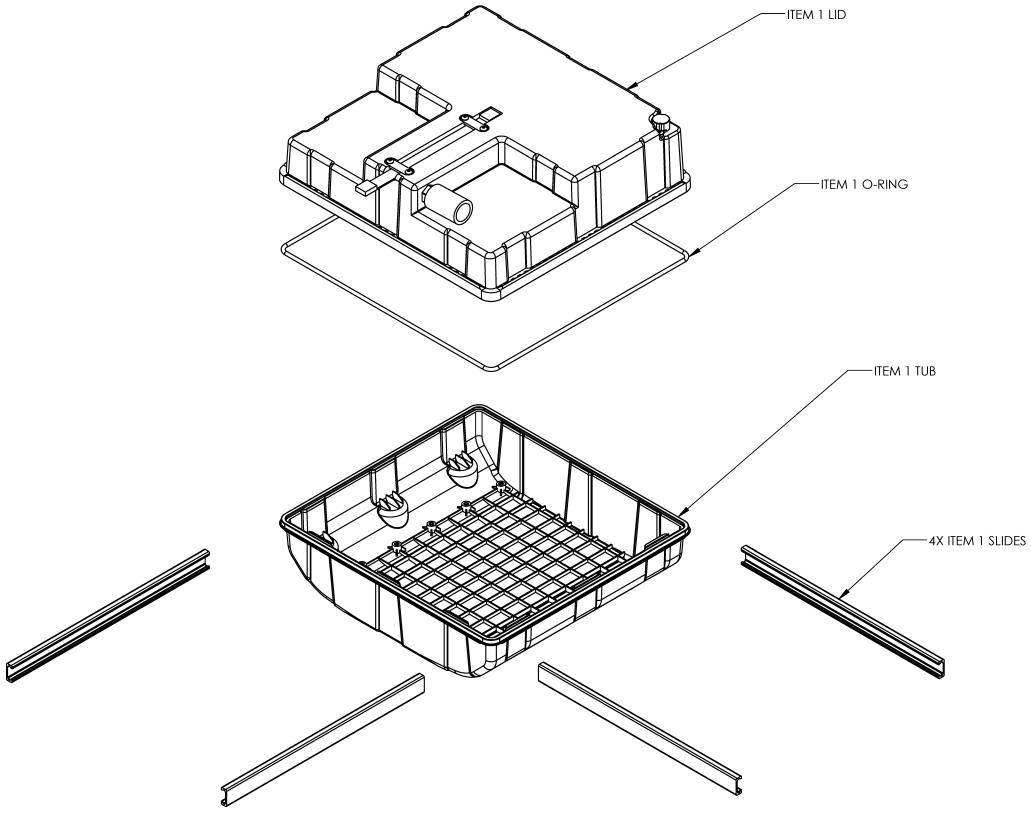
- A two piece Molded Noryl case.
- Water-tight electrical and communications connectors to interface with the CCU.
- A pressurization port for interior pressurization with a non-condensing gas before the case is sealed.
- Brass interior mounting bosses for payload component mounting.

Liquid Robotics Proprietary and Confidential

- Titanium exterior mounting hardware.
- A nylon carrying handle.

The dry box cases are held together by 316 Stainless Steel slide-clamps that maintain pressure on a Parker E 1100 O-ring for a water-tight seal.

Figure 9.10. 3-MPU Assembly



An optional Liquid Robotics electronics package is available for the 3-MPU - for more information see "3-MPU Payload Development Kit" (page 189)).

The Liquid Robotics Auxiliary Power Unit (APU) is a version of a 1-MPU that contains a battery pack and an AMPs power domain. See "Auxiliary Power Unit (APU)" (page 185) for more information.

Note

The CCU enclosure is a version of a 3-MPU.

For general information see:

- "Command and Control Unit (CCU)" (page 8)
- "CCU Weight and Dimensions" (page 9)

For information on payload electrical connections to the CCU see:

- "CCU Electrical" (page 32)

Liquid Robotics Proprietary and Confidential

- "G1, G2, G3 - General Purpose Expansion Port Connectors" (page 44)
- "S1, S2, S3, S4 - Sensor Connectors" (page 47)

Weight of Modular Payload Dry Boxes

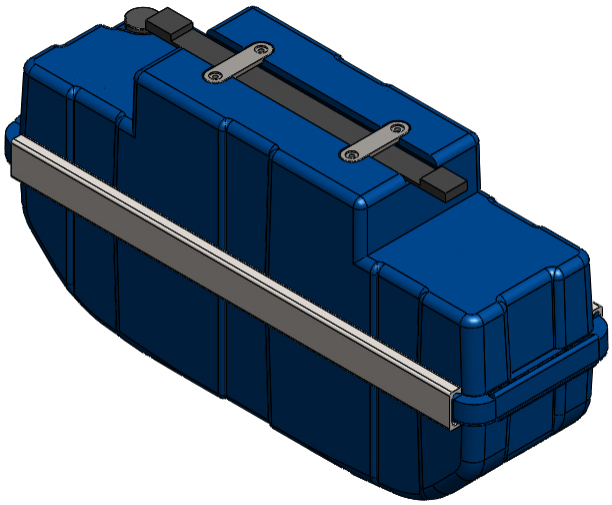
The weight of the Liquid Robotics 1-MPU and 3-MPU is an important consideration in evaluating the feasibility of a payload integration project.

Table 9.3. Weight of Modular Payload Dry Boxes

Modular Dry Box Type	Weight
1-MPU (payload dry box only)	2.27 Kg (5.0 lb)
APU	8.39 Kg (18.5 lb)
3-MPU (payload dry box only)	4.54 Kg (10.0 lb)
3-MPU Payload Development Kit (with standard electronics)	6.12 Kg (13.5 lb)
CCU	17.0 Kg (37.5 lb)

1-MPU Payload Dry Box

Figure 9.11. 1-MPU



Liquid Robotics Proprietary and Confidential

Figure 9.12. 1-MPU Interior Dimensions (mm)

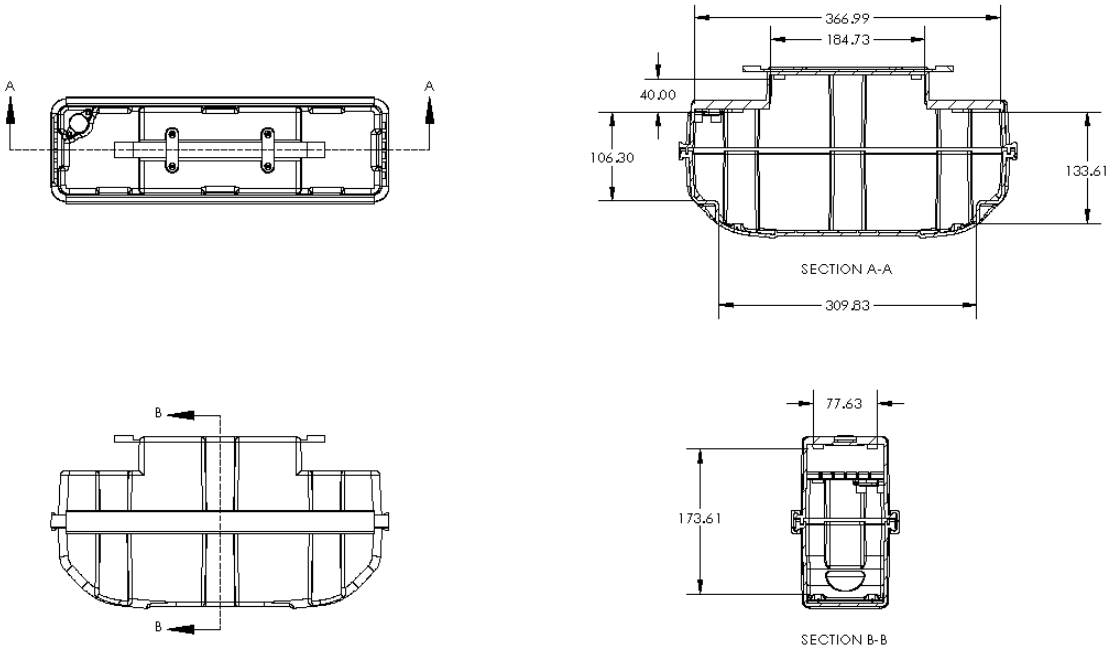
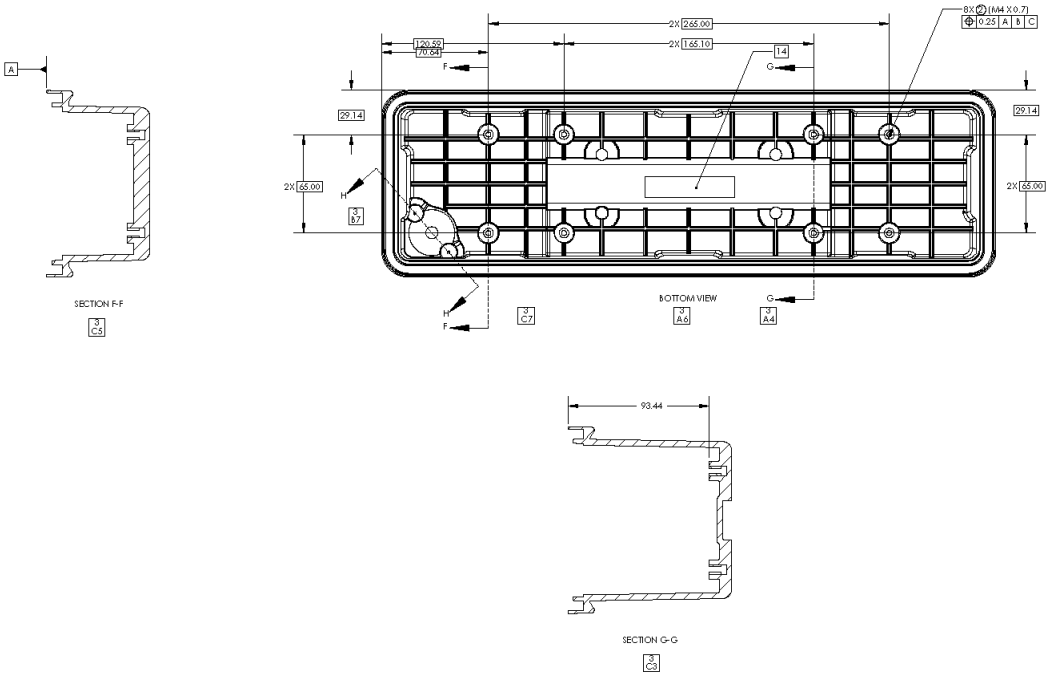


Figure 9.13. 1-MPU Attachment Points (mm)



Liquid Robotics Proprietary and Confidential

3-MPU Payload Dry Box

Figure 9.14. 3-MPU

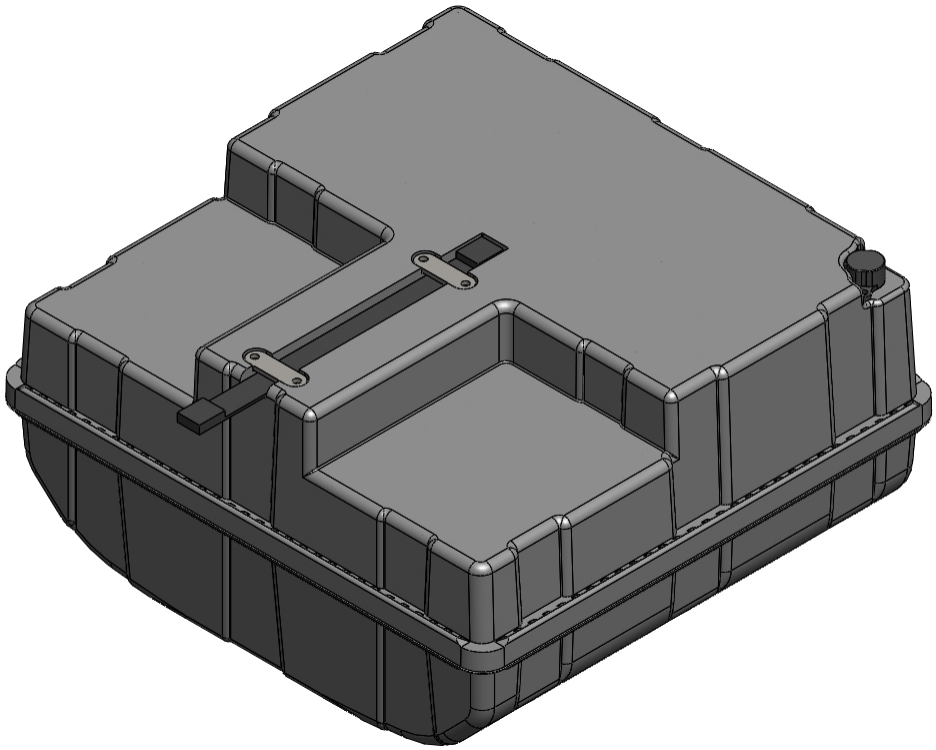


Figure 9.15. 3-MPU Interior Dimensions (mm)

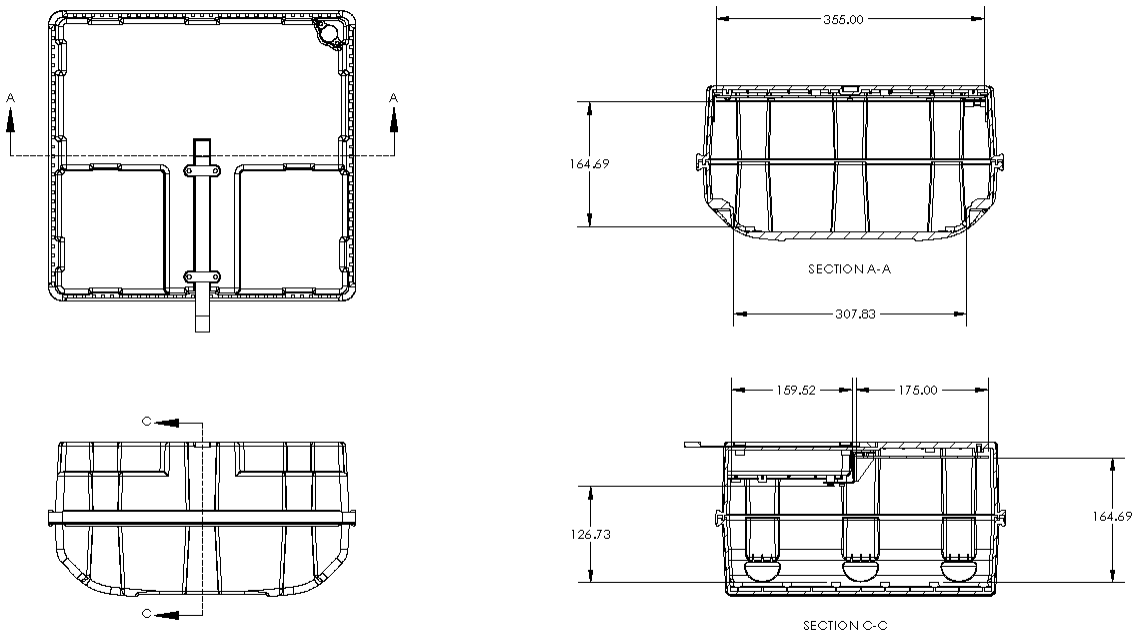
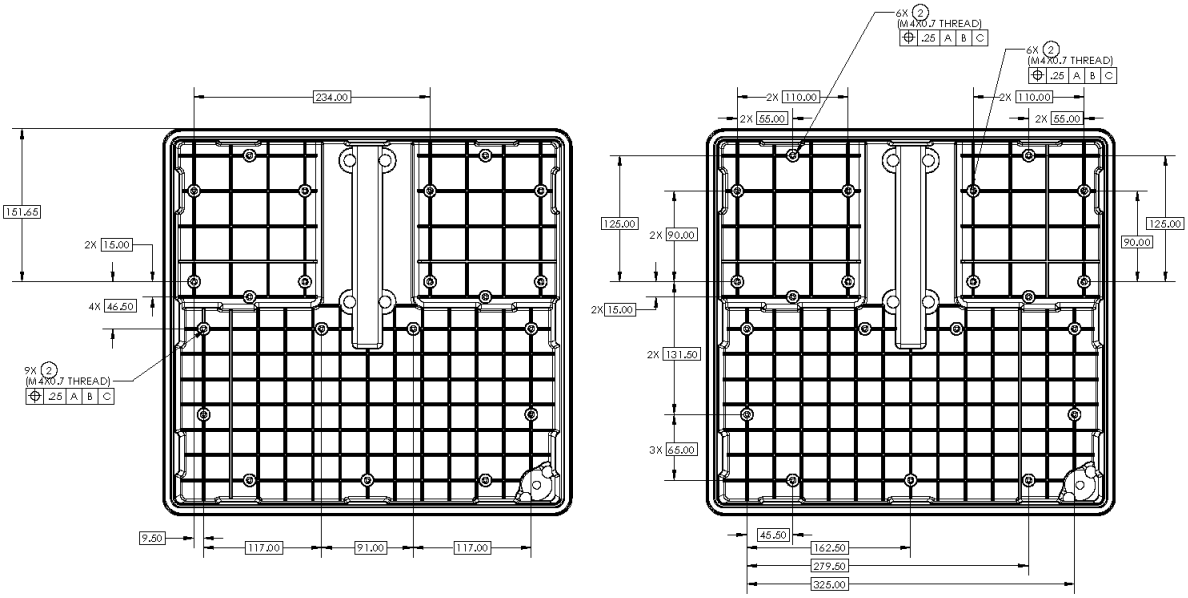


Figure 9.16. 3-MPU Attachment Points (mm)



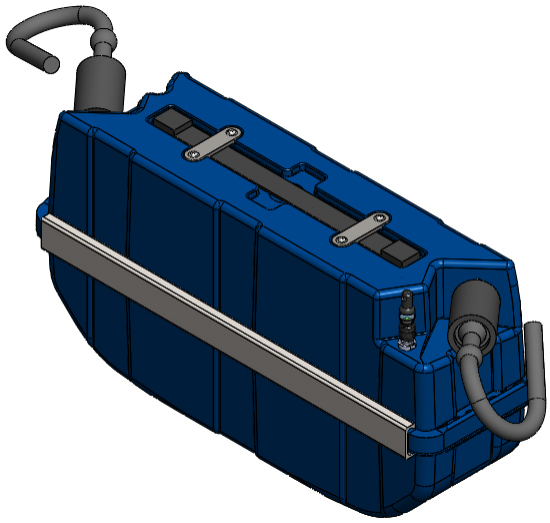
Auxiliary Power Unit (APU)

Adding an APU to a Wave Glider configuration doubles the power storage capacity of the system. Each APU contains the same 980 watt hour battery pack that is included in the CCU.

An APU weighs 8.39 Kg (18.5 lb) and has the same dimensions as a 1-MPU payload dry box.

Liquid Robotics Proprietary and Confidential

Figure 9.17. Auxiliary Power Unit (APU) Dry Box



Multiple APUs can be added to a Wave Glider Float, limited by:

- The total weight of all payload components on the Float.
- The space available in the Float payload bays.
- The center-of-gravity for the Float configuration.

For more information see "Float Payload Considerations" (page 172).

APUs must connect to CCU General Purpose Expansion Ports. Multiple APUs can be daisy-chained together with the first APU in the chain connected directly to a CCU General Purpose Expansion Port. Alternatively, each APU can be connected to a dedicated General Purpose Expansion Port.

For information on the APU electrical connectors and instructions on connecting multiple APUs in a Wave Glider configuration see "APU Electrical Connections" (page 70).

1-MPU Payload Development Kit

The Liquid Robotics 1-MPU Payload Development Kit is designed to simplify payload integration in a Wave Glider SV3 vehicle. The kit includes a 1-MPU Payload Dry Box with the following installed components:

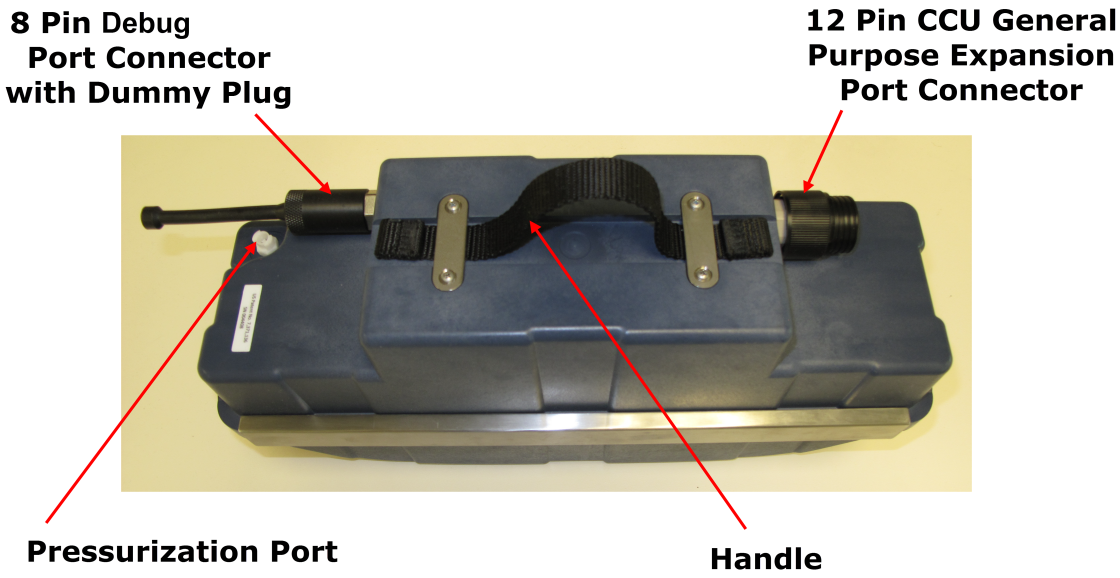
- 12 pin CCU General Purpose Expansion Port connector - for more information see "CCU Expansion Ports" (page 41)
- 8 pin Debug Port connector
- 5 slot AMPS backplane - for more information see "1 MPU Module Carrier" (page 209)
- 1 AMPS Power Domain Controller module - for more information see "Power Domain Controller (PDC) Module " (page 198)

Liquid Robotics Proprietary and Confidential

- 1 AMPS Bridge module - for more information see “Bridge Module” (page 208)
- 1 Buck Boost Regulator (6 x 13.4V) - for more information see “Buck-Boost Regulated Output Module (BBR)” (page 206)
- 1 System Management Computer (SMC) Carrier - for more information see “SMC Carrier” (page 215)
- 1 System Management Computer (SMC - for more information see “Sensor Management Computer (SMC)” (page 217))

The 1-MPU Payload Development Kit weighs 3.40 kg (7.5 lb). See “1-MPU Payload Dry Box” (page 182) for the dimensions of the 1-MPU Payload Dry Box.

Figure 9.18. 1-MPU Payload Development Kit Connectors



Liquid Robotics Proprietary and Confidential

Figure 9.19. 1-MPU Payload Development Kit Electronics A

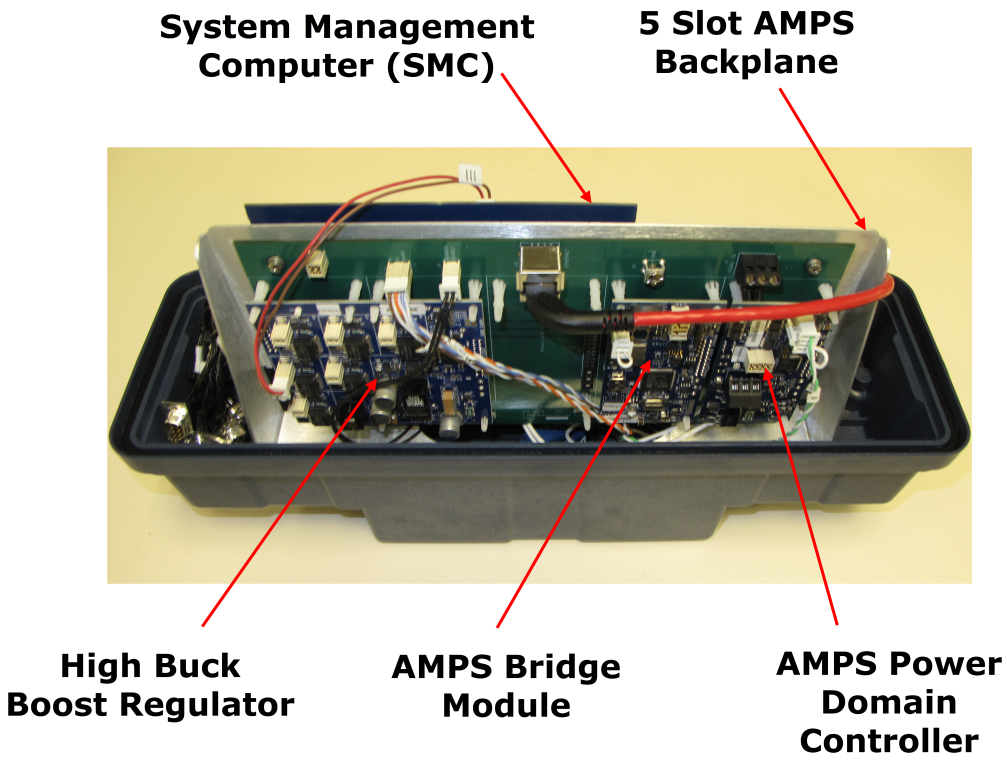


Figure 9.20. 1-MPU Payload Development Kit Electronics B

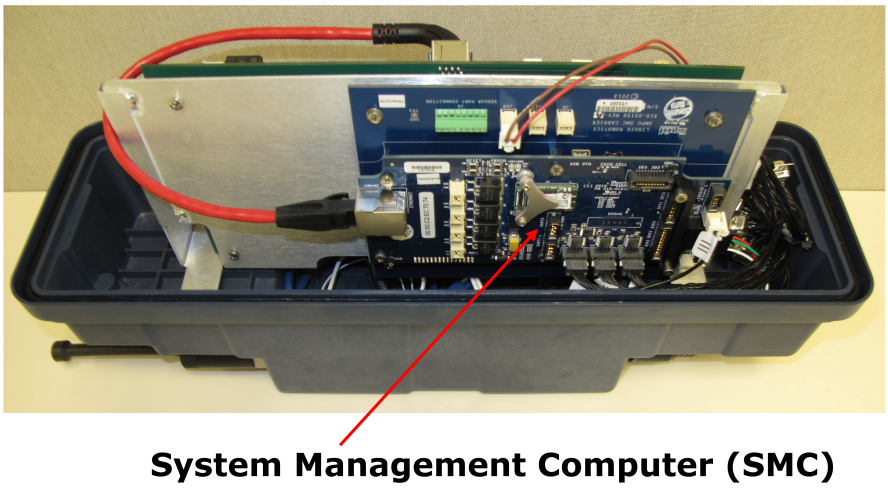
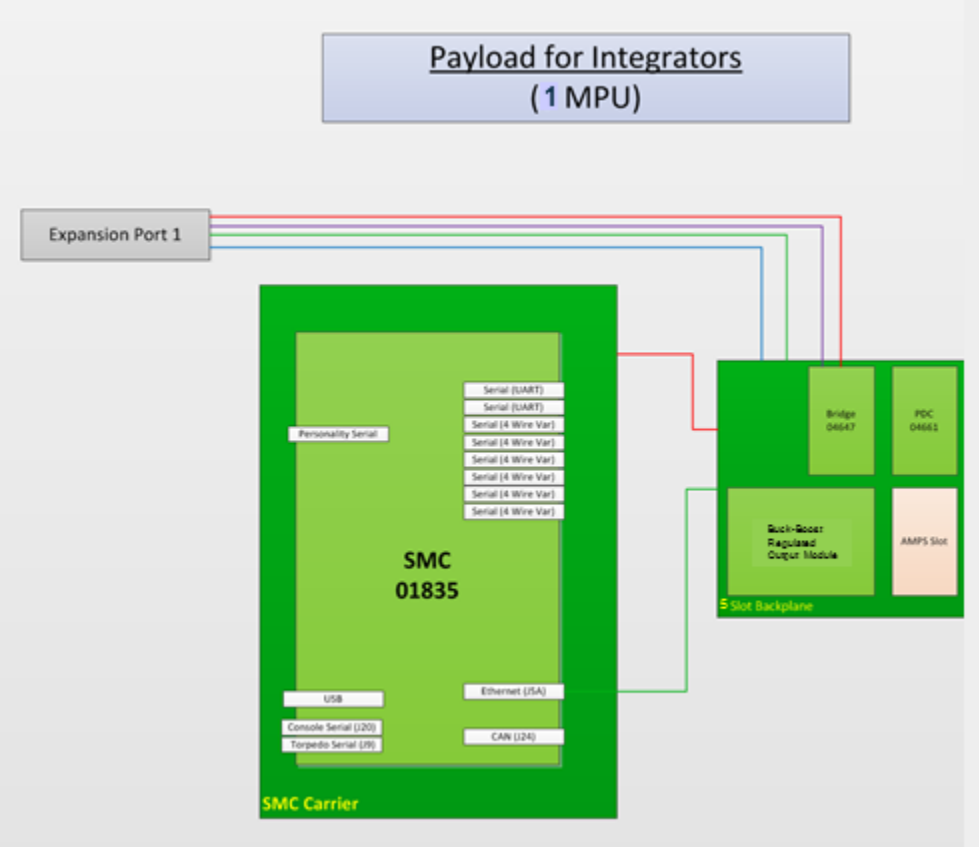


Figure 9.21. 1-MPU Payload Development Kit Electronics Block Diagram



3-MPU Payload Development Kit

The Liquid Robotics 3-MPU Payload Development Kit is designed to simplify payload integration in a Wave Glider SV3 vehicle. The kit includes a 3-MPU Payload Dry Box with the following installed components:

- 12 pin CCU General Purpose Expansion Port connector - for more information see “CCU Expansion Ports” (page 41)
- 8 pin Debug Port connector
- 7 slot AMPS backplane - for more information see “3 MPU Module Carrier” (page 211)
- 1 AMPS Power Domain Controller module - for more information see “Power Domain Controller (PDC) Module ” (page 198)
- 1 AMPS Bridge module - for more information see “Bridge Module” (page 208)
- 1 High Voltage Board (2x high voltage, 6x switched Vcons) - for more information see “High Voltage Module” (page 203)
- 1 System Management Computer (SMC) Carrier - for more information see “SMC Carrier” (page 215)
- 1 System Management Computer (SMC) - for more information see “Sensor Management Computer (SMC)” (page 217)

Liquid Robotics Proprietary and Confidential

The 3-MPU Payload Development Kit weighs 6.12 kg (13.5 lb). See "3-MPU Payload Dry Box" (page 184) for the dimensions of the 3-MPU Payload Dry Box.

Figure 9.22. 3-MPU Payload Development Kit Connectors

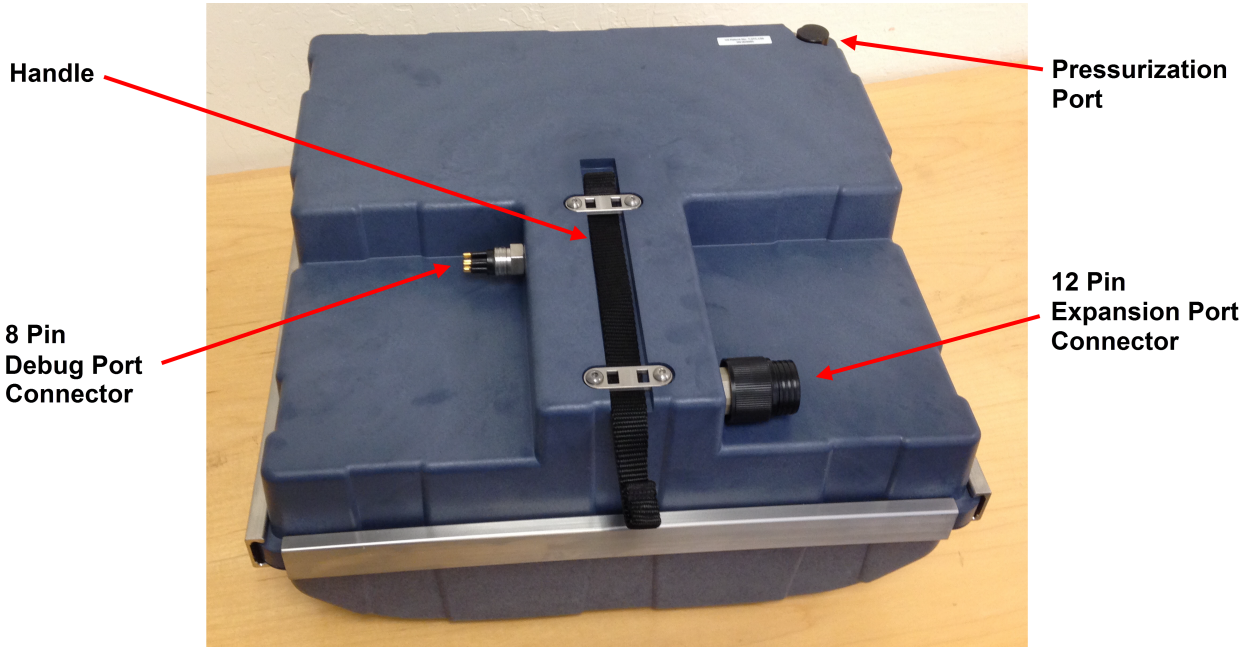


Figure 9.23. 3-MPU Payload Development Kit Electronics

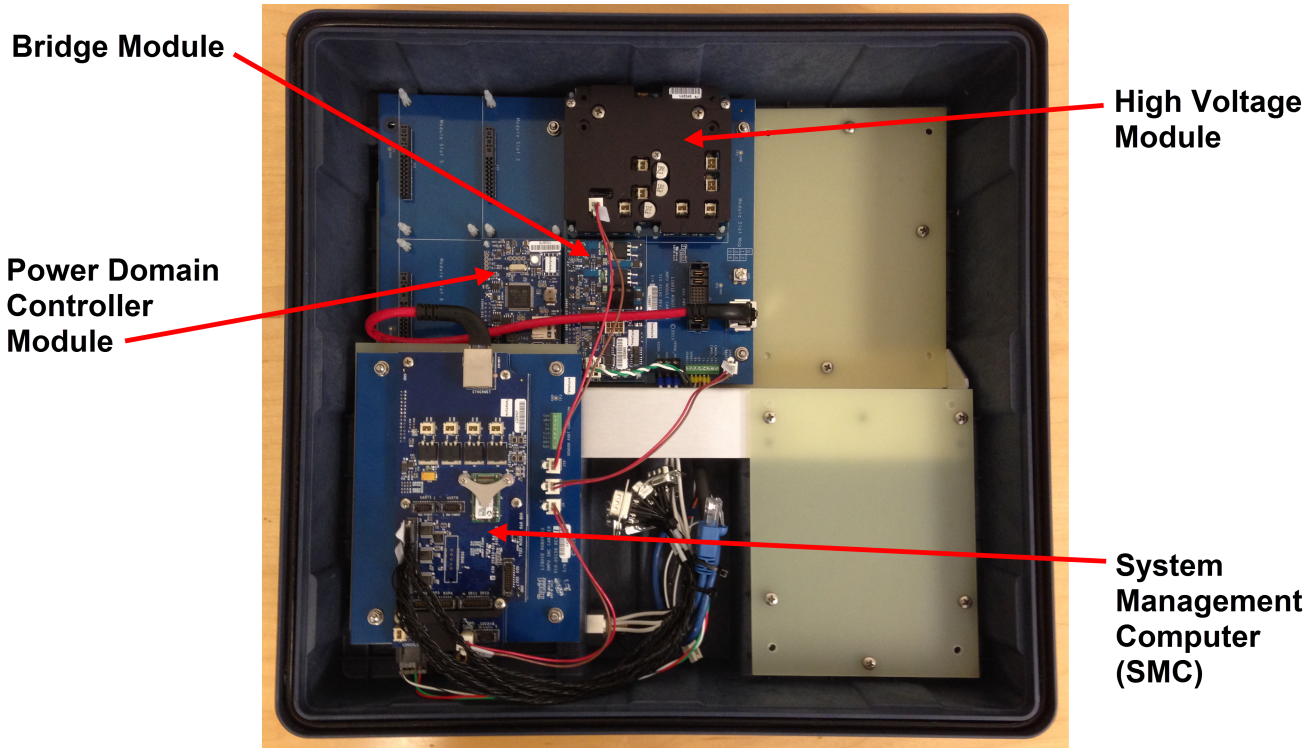
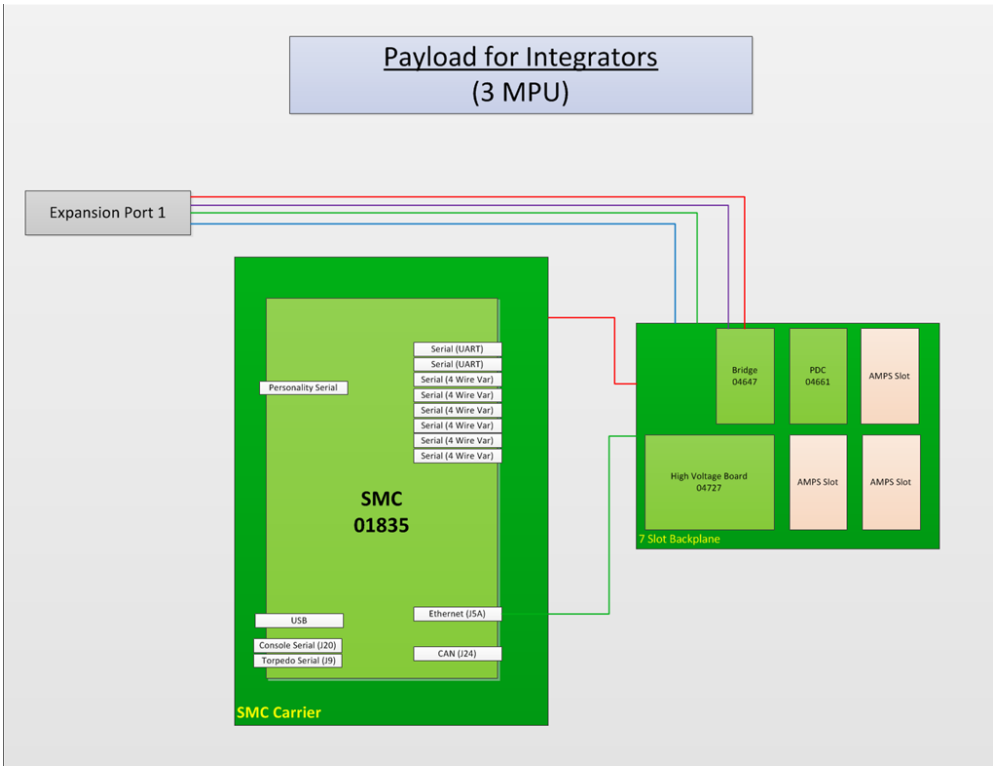


Figure 9.24. 3-MPU Payload Development Kit Electronics Block Diagram



Sub Payload Considerations

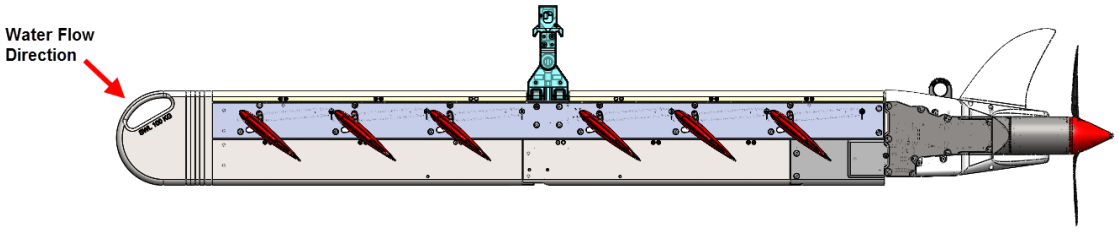
Since the Sub is constantly moving underwater, it is important to consider all drag, balance, and flotation issues when designing a payload.

Drag

To maximize the efficiency of the Wave Glider system, the drag of any payload added to the Sub should be minimized both by reducing the frontal cross-sectional area, and by adding streamlined fairings if necessary.

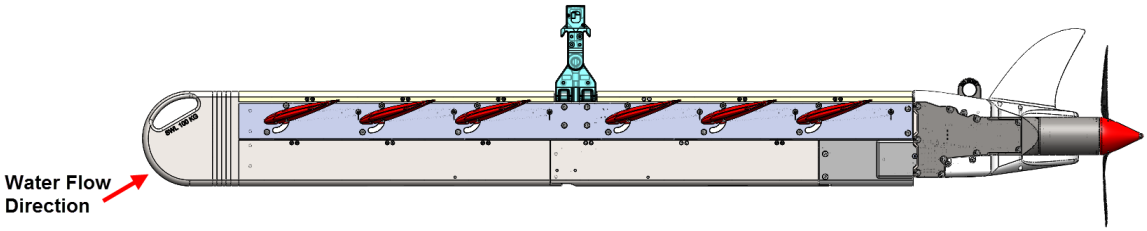
Contrary to what might be assumed, the water flow direction over the Sub is not exactly from front to back. As shown in the two figures below, there is a substantial vertical component to the flow, especially when the Sub is being lifted upward in heavy swells.

Figure 9.25. Sub Moving Up



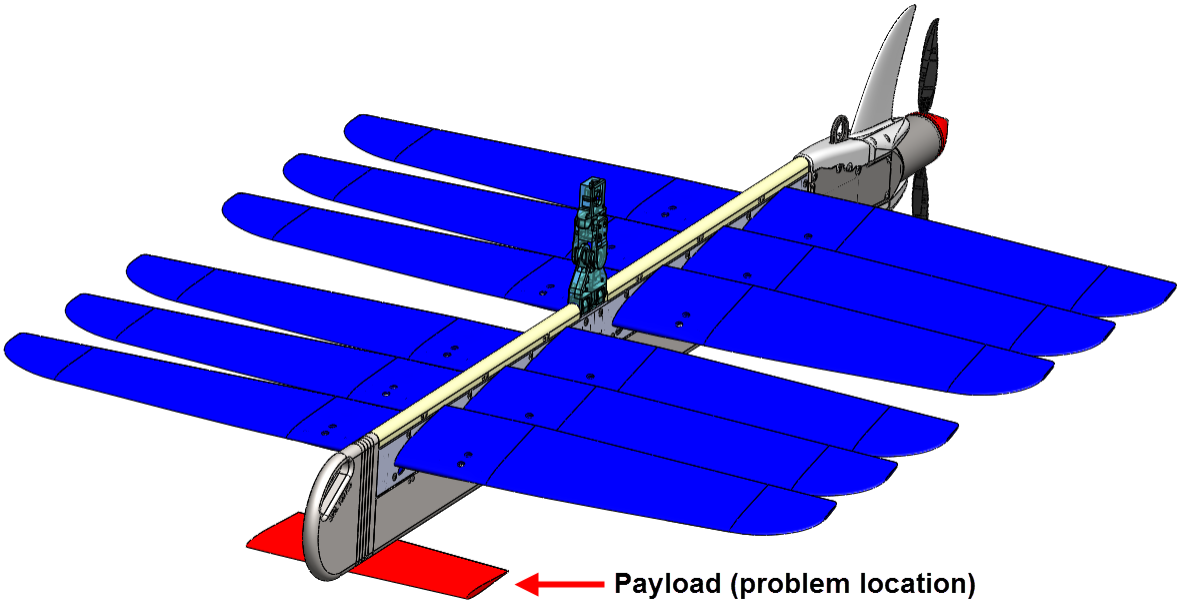
Liquid Robotics Proprietary and Confidential

Figure 9.26. Sub Moving Down



Thus, a seemingly "streamlined" payload, like that shown in Figure 9.27, would create a substantial nose-down force (as well as drag) when the Sub is moving up.

Figure 9.27. Poor Payload Design/Location



Sub Ballast

Ballast weights are attached at the nose of the Sub frame as shown in Figure 9.28. As payload is added to the Sub, these weights can be removed or replaced with other weights to adjust the balance and buoyancy.

Liquid Robotics Proprietary and Confidential

Figure 9.28. Sub Ballast

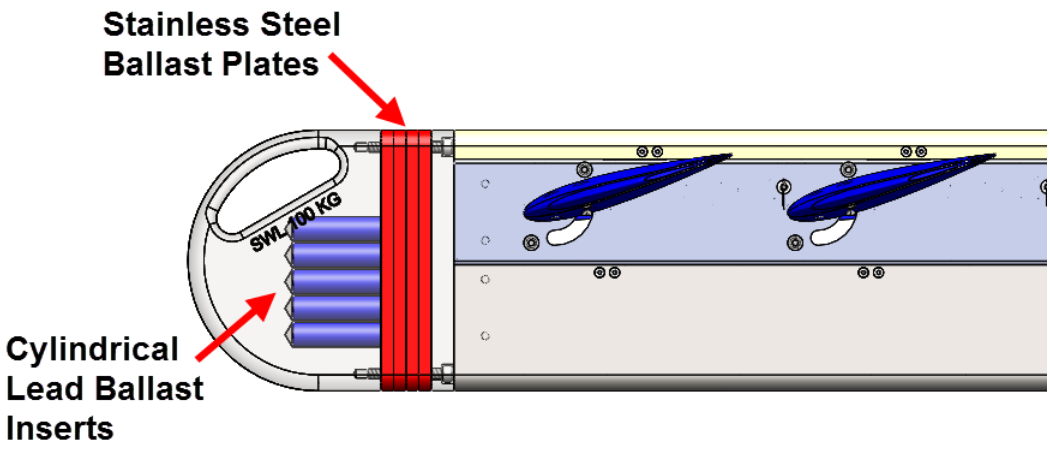
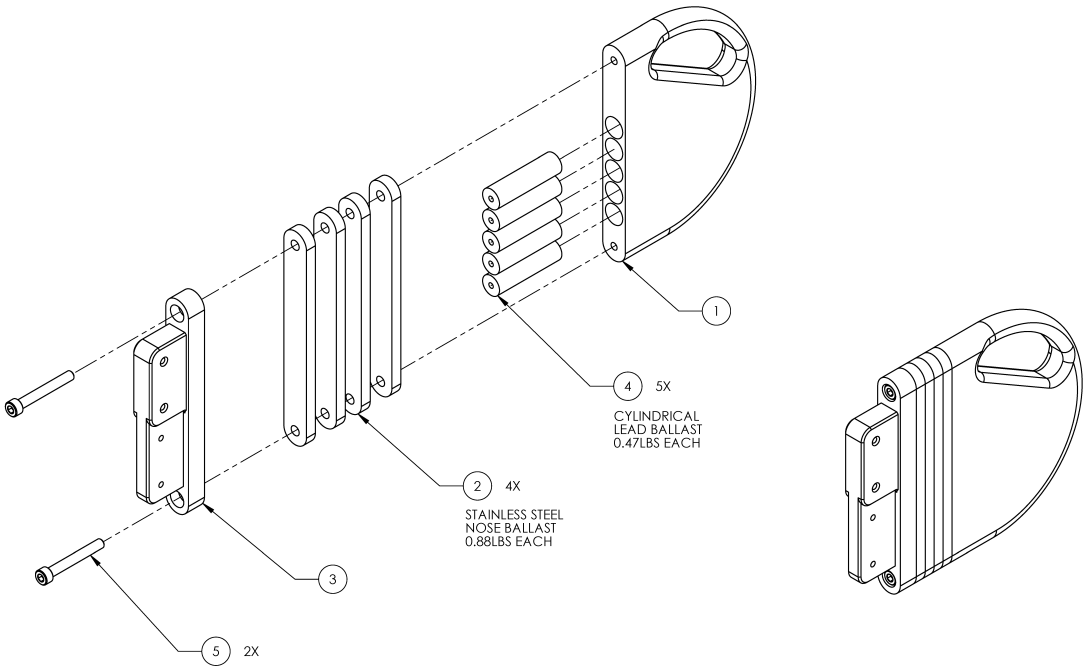


Figure 9.29. Sub Nose Assembly with Ballast

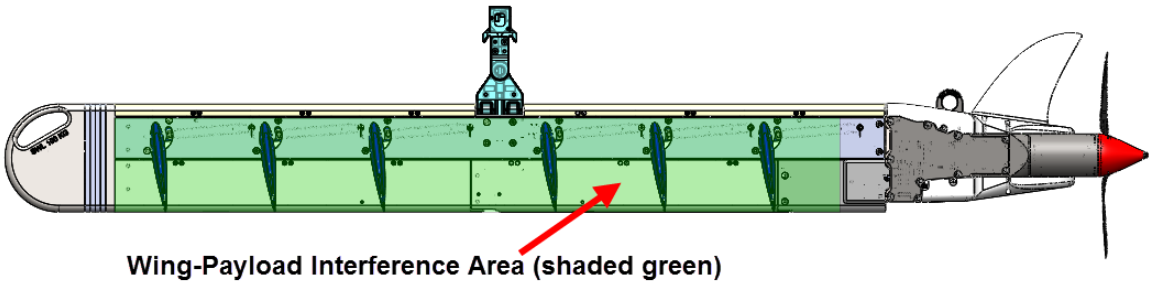


Wing-Payload Interference

Since the Sub wings can rotate upward or downward almost 90° to the vertical position, any payload (or payload cover) mounted within the main frame of the Sub has to accommodate clearance of the wings in their fully rotated position as illustrated in the figure below.

Liquid Robotics Proprietary and Confidential

Figure 9.30. Wing-Payload Interference Area



Balance and Flotation

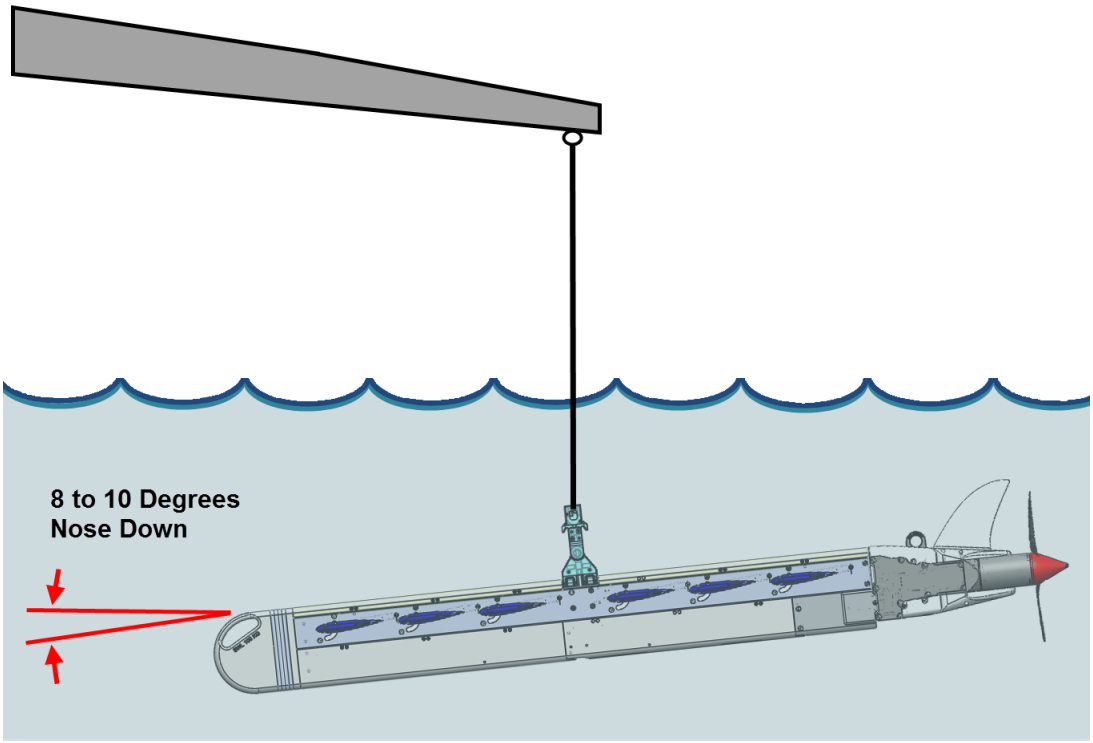
A Sub with an added payload needs to be balanced about the Umbilical pivot point. Since the Sub is submerged, the balance needs to be established underwater, to account for both the flotation and the weight of the payload.

The best way to compute the flotation and balance of the Sub is to support it by the Umbilical attach point and check the deck angle while the Sub is submerged. This procedure is more accurate if performed in salt water, but it can be done in fresh water.

The Sub should be either level, or slightly nose-down. The maximum nose-down angle is shown in the figure below.

The Sub should never be ballasted nose-up.

Figure 9.31. Checking Sub Underwater Balance



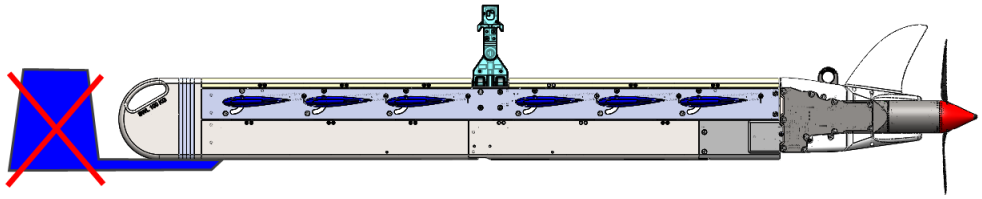
Liquid Robotics Proprietary and Confidential

Poor Sail Area Locations

A Sub payload with a large sail area can be a problem if the payload is mounted forward or aft of the Sub, as shown in the figures below.

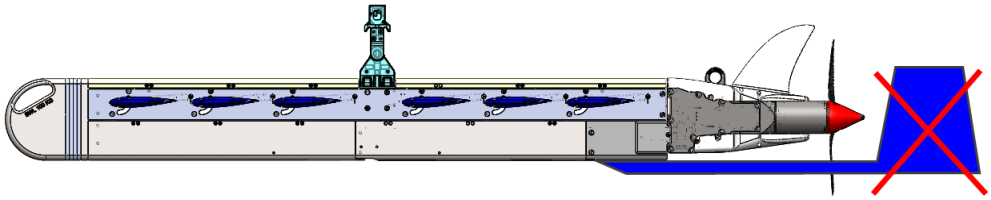
A large sail area in front of the Sub can destabilize and make it over-sensitive to the rudder.

Figure 9.32. Excessive Forward Payload Sail Area



Conversely, a large sail area aft of the Sub can over-stabilize and reduce the effectiveness of the rudder.

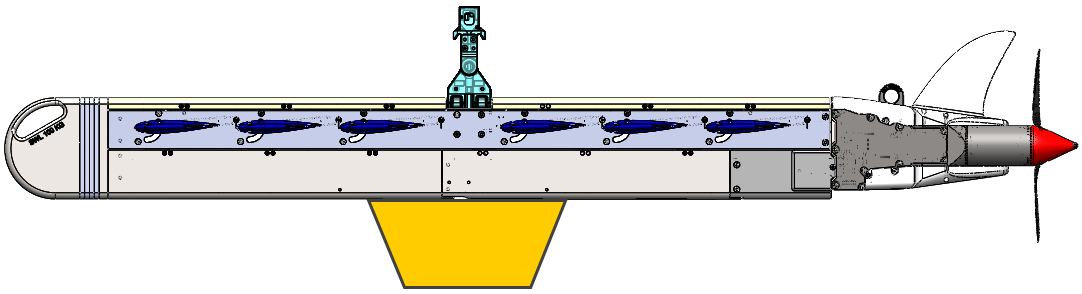
Figure 9.33. Excessive Aft Payload Sail Area



Acceptable Sail Area Locations

Generally, a reasonably-sized payload sail area located below the Umbilical attach point should not materially affect stability, though there can be a significant increase in drag for payloads mounted outside the frame.

Figure 9.34. Acceptable Payload Sail Area Location



Trailing Wires

Trailing wires behind the Sub are generally not recommended, as they can potentially foul the Thrudder. However, if a trailing wire is necessary, then the wire should be mounted near the center of the Sub,

Liquid Robotics Proprietary and Confidential

and not trail off of the back. In addition to fouling, rear trailing wires can have the same over-stabilizing effect as a large sail area at the aft end of the Sub.

Figure 9.35. Poor Trailing Wire Location (Tow Point)

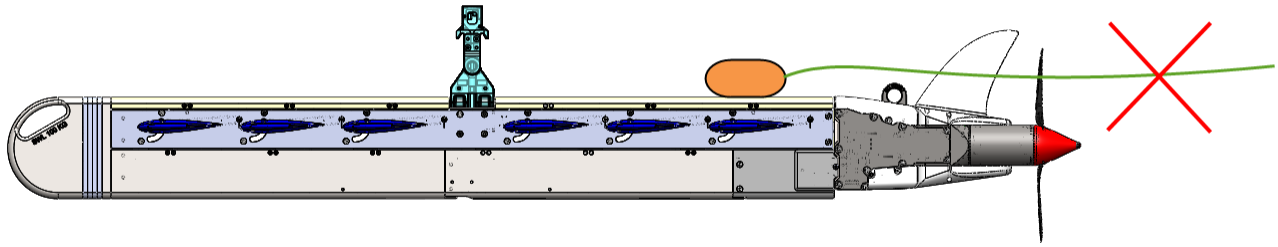
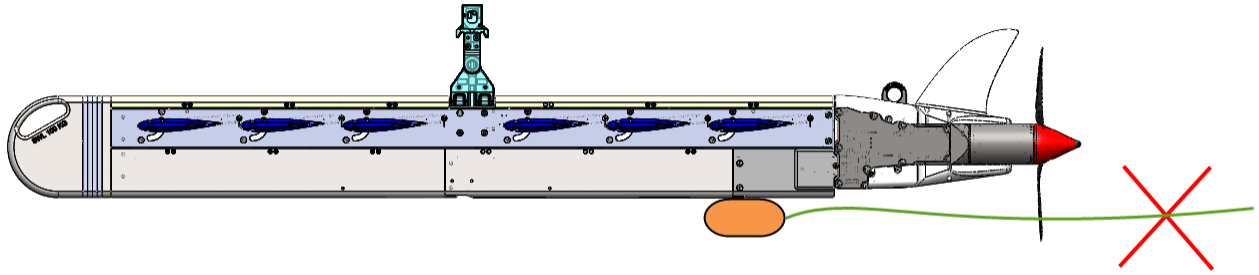
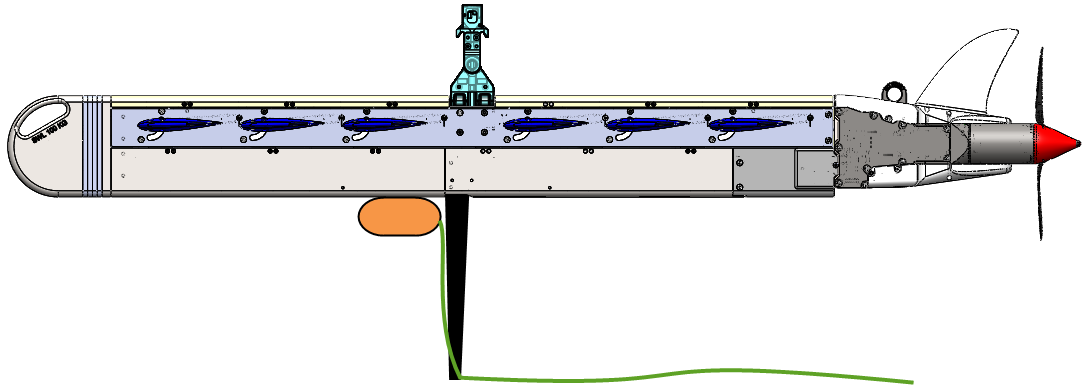


Figure 9.36. Poor Trailing Wire Location (Tow Point)



The best location for a trailing wire mounting is on a pylon directly below the Umbilical attach point, as shown in the figure below.

Figure 9.37. Acceptable Trailing Wire Location (Tow Point)



Keep in mind that the Sub is continually moving up and down while moving forward. Any trailing wire must be short enough and mounted so it cannot foul the Thrudder, regardless of the angle of the wire.

Liquid Robotics Proprietary and Confidential

Towfish Payload Considerations

There are a wide range of variables that impact Towfish payload integration. Contact Liquid Robotics for help evaluating the feasibility of a Towfish payload integration project.

For information on Towfish electrical connections, see “Thrudder Module Expansion Port Connector” (page 65).

For information on the Towfish Battery Charger, see Appendix D: *Towfish Battery Charger Technical Specifications*.

Chapter 10. Payload Development - Hardware Environment

The SV3 payload development hardware environment is defined by the AMPS backplane, the various AMPS modules, and the System Management Computer (SMC).

- For information on AMPS and the AMPS backplane see “Adaptable Modular Power System (AMPS)” (page 38).
- For information on AMPS modules see “AMPS Modules” (page 198).
- For information on the SMC see “Sensor Management Computer (SMC)” (page 217).

AMPS Modules

Each AMPS power domain includes a Power Domain Controller (PDC) along with other AMPS modules depending on the power management requirements and the configuration.

Power Domain Controller (PDC) Module

The Power Domain Controller module (PDC) is managed by higher level systems through the CCU and also provides those systems with AMPS system status. There is one PDC module in each AMPS power domain. The PDC module communicates with the Vehicle Management Computer in the CCU over CAN to monitor and control all of the other AMPS modules in a power domain. The PDC also includes pressure and temperature sensors to monitor environmental conditions in the dry box that houses the AMPS domain.

Module Data - The following data will be collected:

- Pressure
- Temperature
- A log of communications to and from AMPS modules in the power domain that the PDC controls (for troubleshooting purposes).
- A file system that contains the current firmware versions for all of the AMPS modules in the power domain that the PDC controls.

Liquid Robotics Proprietary and Confidential

Figure 10.1. AMPS Power Domain Controller Module (PDC)

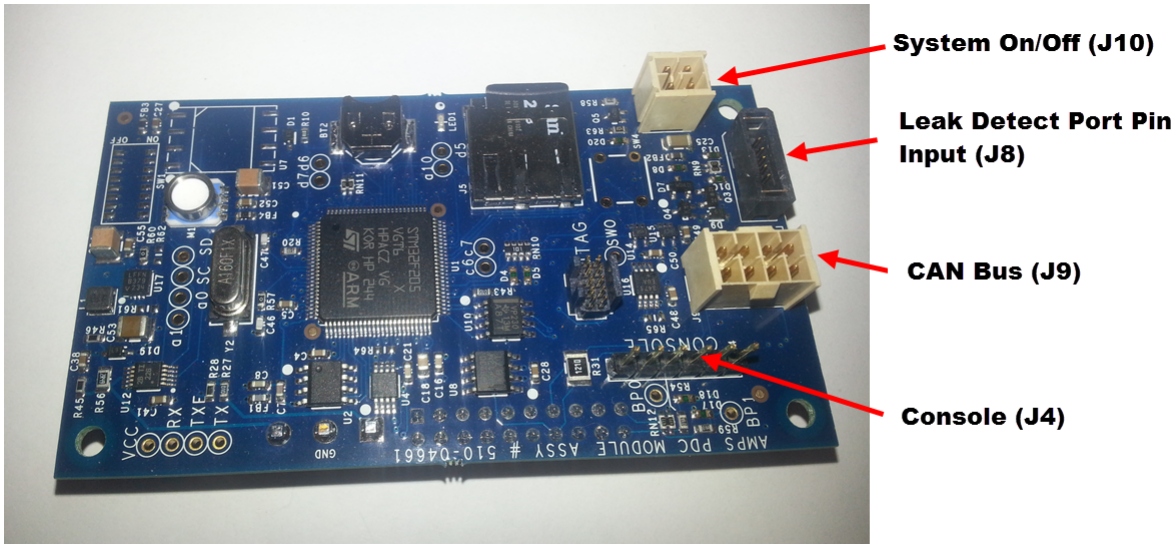


Table 10.1. System On/Off (J10)

Pin	Signal	Description
1	PWR_SW_IN	System power switch. =low means system power on.
2	GROUND	Signal ground.
3	GROUND	Signal ground.
4	EXT_LED+V	Power source for external LED. Controlled by PDC processor.

When the PWR_SW_IN signal is closed (shorted to ground), and the domain contains one or more power sources (batteries), they will commence providing power. The PDC will detect when the switch is opened, and it will control the power down programmatically. There may be a noticeable delay between when the switch is thrown and when the power goes off.

The EXT_LED+V signal is also controlled by the PDC. It indicates that the PDC is operating by its blink rate.

Table 10.2. Leak Detect Port Pin Input (J8) - currently not used

Pin	Signal	Description
1	GROUND	Signal to ground.
2	LEAK1	Input to GPIO on processor (ADC capable).
3	NO CONNECT	
4	GROUND	Signal ground.
5	LEAK2	Input to GPIO on processor (ADC capable).

The leak detect signals are wired directly to the processor, protected by ESB suppressors. Currently there is no firmware associated with these pins.

Liquid Robotics Proprietary and Confidential

Table 10.3. CAN Bus (J9)

Pin	Signal	Description
1	CANH_PRI	CAN bus – Recessive signal. Primary bus.
2	CANL_PRI	CAN bus – Dominant signal. Primary bus.
3	GROUND	Signal ground.
4	CANPRI_DETn	CAN bus detect – low active. Primary bus.
5	CANH_SEC	CAN bus – Recessive signal. Secondary bus – not used.
6	CANL_SEC	CAN bus – Dominant signal. Secondary bus – not used.
7	GROUND	Signal ground.
8	CANSEC_DETn	CAN bus detect – low active. Secondary bus – not used.

Only the primary bus of this dual CAN bus connector (running 125kbps) is used by the firmware at this time. The bus detect signal is tied to ground within the connector, thereby indicating its presence. Other than displaying a message on the console port, there is no other use for, or consequence of, this signal.

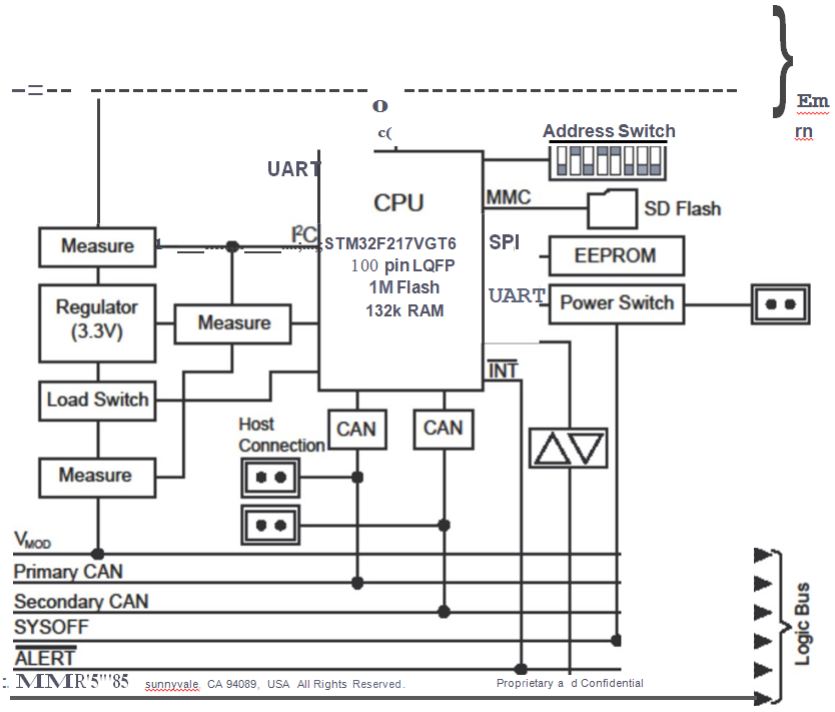
Table 10.4. Console (J4)

Pin	Signal	Description
1	GROUND	Signal ground.
2	CTS	Clear to Send – not used.
3	VCC	+3.3VDC power – no connect.
4	TXD	Transmit data (from board).
5	RXD	Receive data (into board).
6	RTS	Ready to Send – not used.

This console connector provides a window into the PDC processor. It is a CMOS-level (3.3V) serial port that can be used to observe what is happening within the PDC. **It can also be used to change processor settings - proceed with caution.**

Liquid Robotics Proprietary and Confidential

Figure 10.2. PDC Electrical Diagram



Battery Charger Module

The battery charger module is designed to control the charge and discharge of two Inspired Energy SMBus SMART batteries. It contains a feedback circuit to optimize charging, and it will report on the state of the batteries. Hot-swapping of batteries is not supported.

Table 10.5. Electrical Specifications

Parameter	Description	Maximum
I_{CONS}	Maximum current delivered onto VCONS from batteries and VPROD [1]	10 A
C_{VCONS}	Maximum capacitance on VCONS	2000 uF
I_{CHG}	Maximum charging current	4 A

[1] This current limit is not enforced by the hardware. It must be considered and respected when the host controls the power system.

Figure 10.3. Battery Charger Module

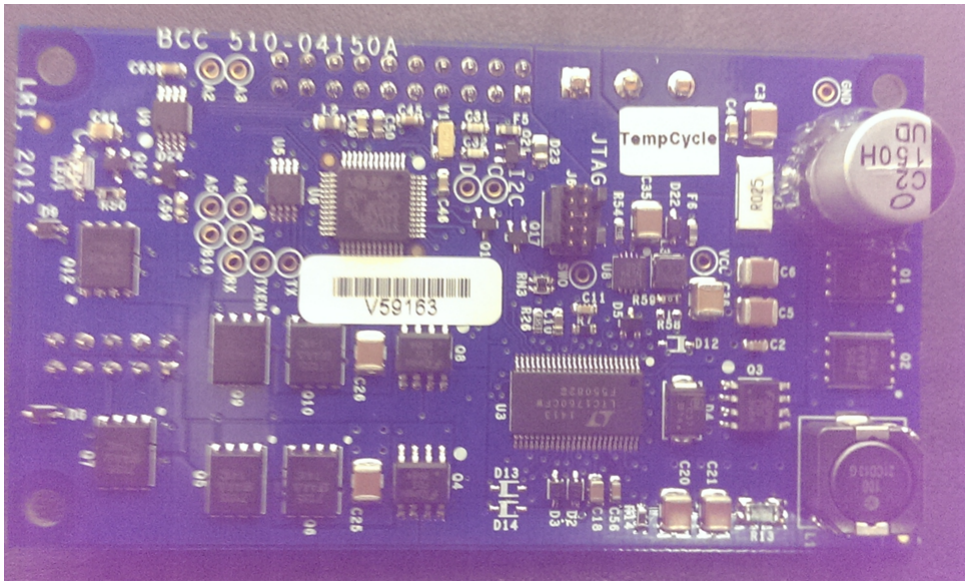
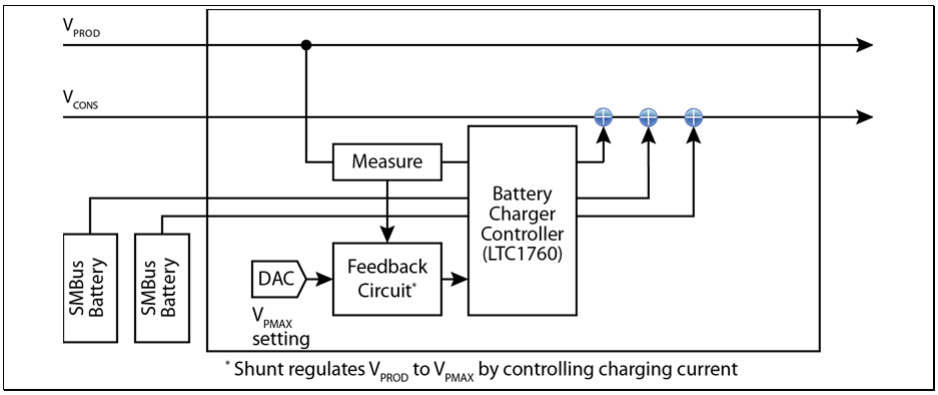


Figure 10.4. Battery Charger Module Electrical Diagram



Solar Panel Input Module

A Solar Panel Input module can receive power from one or two solar panels. The output of each solar panel is connected to the VPROD bus that supplies power to replenish the batteries and operate the system. The Solar Input module monitors the energy collected and also provides a way to disconnect the Solar Panel inputs from the VPROD bus, if there is a cable fault.

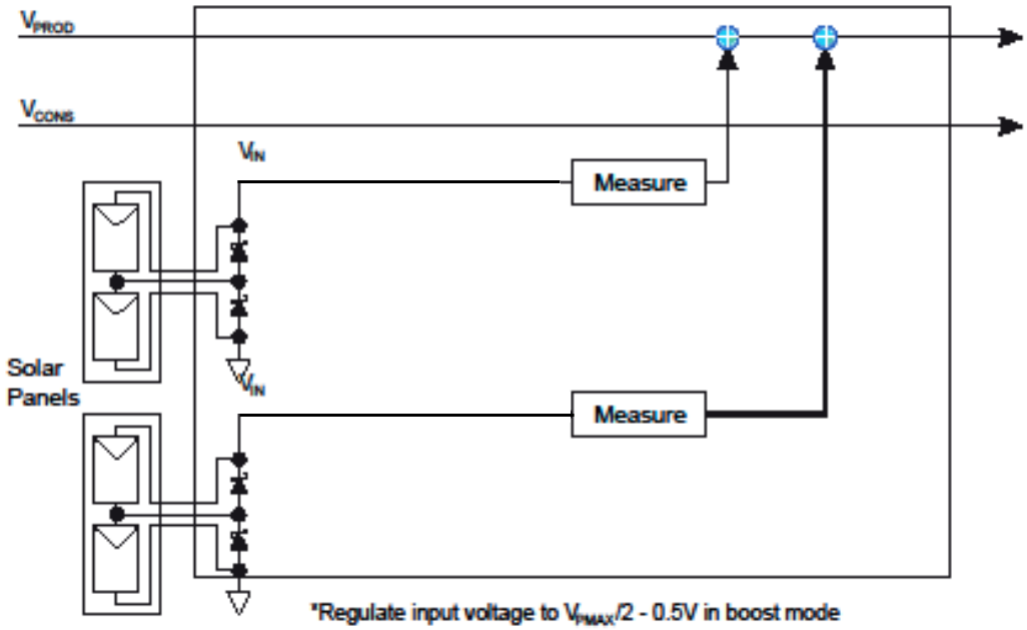
Module Data - Solar Input modules return the following data:

- Total energy collected.
- Maximum, average and standard deviation of power collected (over a 10 minute non-overlapping sampling period).

Liquid Robotics Proprietary and Confidential

Figure 10.5. Solar Panel Input Module

Figure 10.6. Solar Panel Input Module Electrical Diagram



High Voltage Module

The AMPS High Voltage module (HV) is a “double wide” power supply module. It has 8 outputs; 2 outputs are programmable high voltage outputs and 6 outputs are a switchable version of Vconsumer.

The output voltage for the two high voltage power supplies is programmable using switches on the High Voltage module. There is only one set of switches that sets the output for both power supplies - they are not independently programmable. There are 4 possible settings:

Liquid Robotics Proprietary and Confidential

- +24V @ 4.2A
- +28V @ 4.2A
- +44V @ 2.045A
- +48V @ 1.875A

Each HV output is limited by two factors:

- Current Limit - The HV output has a built-in current limit (nominally 4.2A) that will fault if the current is exceeded for more than several milliseconds. Loads with large inrush (such as loads with large capacitance) may trigger this protection. This circuit is identical to one used on the “Buck-Boost Regulated Output Module (BBR)” (page 206).
- Power Limit - The power converter has a limit of approximately 90watts, depending on input voltage, output voltage, and ambient temperature. Higher boost ratios (low Vconsumer voltage, high output voltage settings) result in the lowest output power capability. Higher output power may be possible with additional characterization testing and better cooling.

Contact Liquid Robotics if you need HV outputs that are outside of these limits.

The current sense for HV outputs power measurement is relatively low. The sense resistor is nominally 12milliohms. Devices connected to the HV output may have a ground offset proportionate to their load current. For many devices, the offset is relatively small and should not be a concern. However, for some devices, especially those with a large dynamic load current, this offset effectively results in a common mode noise "on" being seen by other signals (for example, communication signals). This is something to keep in mind when connecting devices that have data interfaces – preferred interfaces will be those that use differential signaling with good common mode rejection. If employing a device with a single ended interface, in the worst case you may need to employ a galvanic isolated transceiver to avert noise-related communication issues. Avoid extra ground connections to devices connected to the HV ports. If an extra ground is connected, power measurement at the HV port may be compromised. For example, some current may return through the extra ground, instead of through the HV port connection, rendering the power measurements at the HV port inaccurate.

Each of the 8 ports on the HV module has an electronic fuse/current limit. Current will be limited to the nominal 4.2A current limit of each port by the E-fuse circuit. HV outputs can have additional restrictions on maximum current, depending on the voltage setting, input voltage, and ambient temperature. If the load demands a current in excess of the current limit, the current to the load will be throttled (restricted) by the current limit circuit to the maximum nominal current limit. During this time, the internal circuitry of the current limit circuit will dissipate a significant amount of power (current is flowing to a load at a reduced voltage and voltage is dropped across the circuitry of the current limit, resulting in high power dissipation). When current limiting is in effect, a nominal 160mS timer is started. If current limiting continues, the timer will timeout, the output will be turned off, and an error condition will be flagged. When switching on power to a capacitive load, the internal circuitry will limit its own power dissipation to a fixed nominal 29W. Since the output voltage is starting at 0V (worst-case), the current will be limited to a lower value = 29W/14.4Vnominal ~ = 2A. As the load capacitance charges, voltage across the internal switch is reduced allowing the current to rise while the internal power dissipation is maintained @ 29W. When the capacitance is charged to slightly over half the supply voltage, the current will have risen to the nominal 4.2A current limit and will remain at this level until either the timeout is reached (and the output is turned off) or the load capacitance has fully charged to the supply voltage (thereby reducing the current to the level required by the load itself).

Most electronic loads with typical input power bypass capacitance will operate satisfactorily with the E-fuse. If there is a need to power a load with significant inrush or a load that is expected to draw a current close to the E-fuse limit, then it would be best to perform advance testing with the load to verify compatibility with the E-fuse circuit.

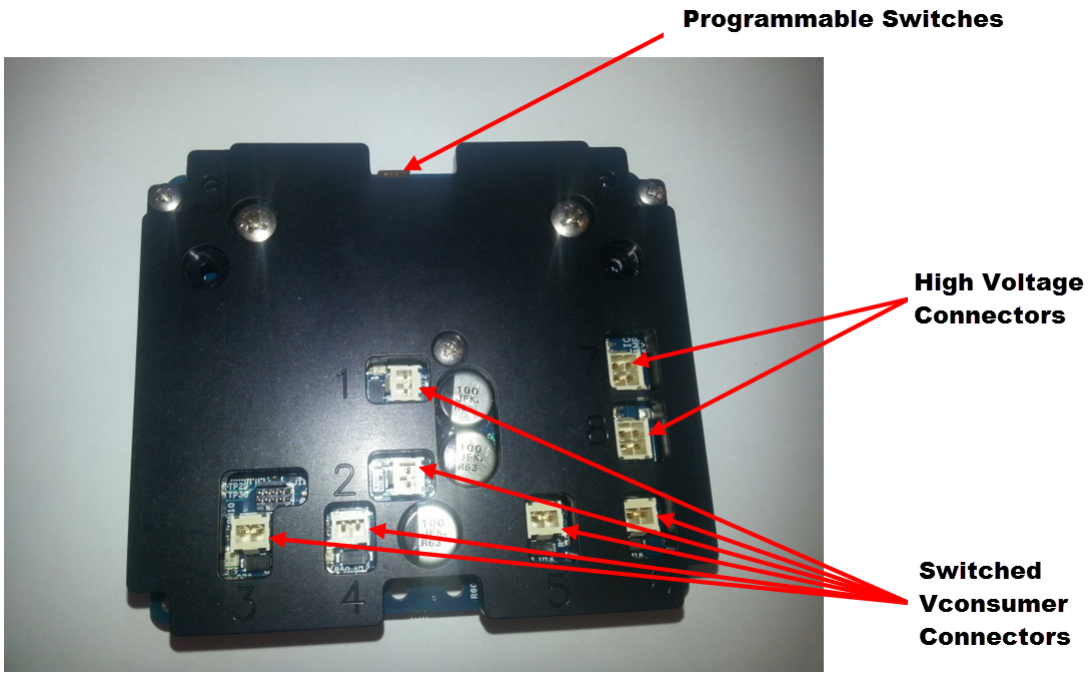
Liquid Robotics Proprietary and Confidential

The AMPS High Voltage module is controlled by RS485 signaling employing the AMPS protocol. When connected to an AMPS host backplane, connectivity is achieved using a Samtec HPMC-110-03-L-D-03 according to the table below:

Pin		Pin	Signal
P3	VCONS		
P2	GND		
P1	NC	NC	NC
2	MA1	1	MA0
4	MA3	3	MA2
6	MA5	5	MA4
8	MA7	7	MA6
10	SDA_A	9	SDAT_B
12	ALERTn	11	NC
14	NC	13	NC
16	NC	15	NC
18	RSTA	17	RST_B
20	NC	19	NC

All 8 outputs (6 switched Vconsumer ports and 2 HV ports) are available through a feature connector on the backplane side of the HV module. The HV ports on the feature connector do not include a separate ground return pin for each output so the HV power provided using the feature connector *will not* be measured by the power measurement circuitry. The 6 switched Vconsumer ports will provide measured power through the feature connector.

Figure 10.7. High Voltage Module



Liquid Robotics Proprietary and Confidential

Table 10.6. High Voltage Connector Pin Out (4-pin)

Pin	Signal	Description
1	24-48V	High Voltage Output
2	24-48V	High Voltage Output
3	Ground	Return Path
4	Ground	Return Path

Mating connector: Samtec p/n IPD1-02-D-K

Table 10.7. Switched Vconsumer Pin Out (2-pin)

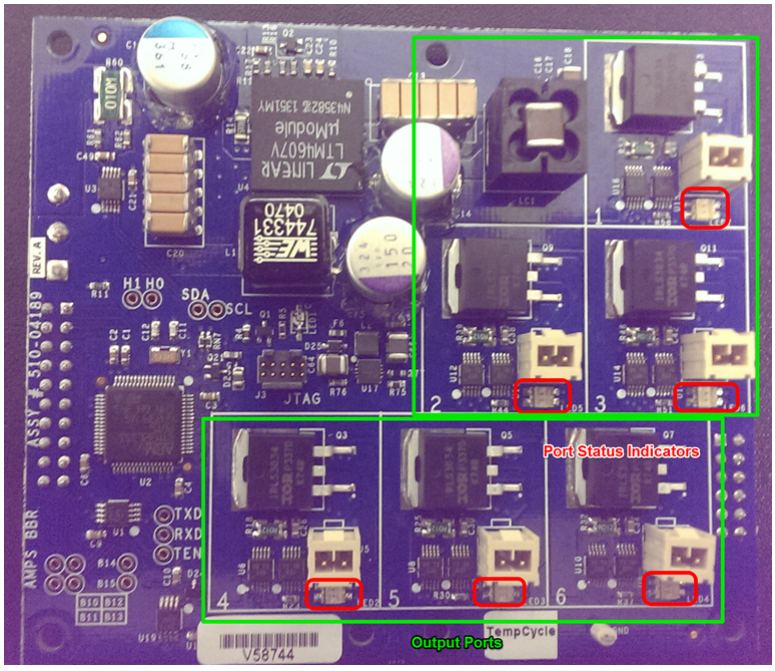
Pin	Signal	Description
1	Vconsumer	3A Limit
2	Ground	Return Path

Mating connector: Samtec p/n IPD1-02-S

Buck-Boost Regulated Output Module (BBR)

The AMPS Buck Boost regulated output module (BBR) is a “double wide” power supply module, taking up two slots on the AMPS backplane. It has 6 outputs each regulated to 13.4 VDC. Each output is current limited to approximately 3 Amps and individually toggled with software. The 6 output ports are outlined in red in Figure 10.8.

Figure 10.8. AMPS Buck-Boost Regulated Output Module (BBR)



Liquid Robotics Proprietary and Confidential

Table 10.8. Electrical Specifications

Parameter	Description	Minimum	Typical	Maximum	Units
V_{CONS}	Input voltage range	12		24	V
I_{OUTN}	Output current limit for a single load output			3.5	A
I_{OUT}	Total output current for all loads			5	A
C_{LOAD}	Output capacitance for a single load			50,000	uF
V_{OUT}	Output voltage	13.0	13.4	13.8	V
V_{NOISE}	Residual peak-to-peak noise at a regulated output			100	mV
η_{REG}	Regulator conversion efficiency	90			%
t_{SC}	Load short circuit duration				s

Port Status LED's

Each BBR Output port has an accompanying status LED indicator. Under **Normal operation** the status LED is **GREEN**. If a **FAULT** has occurred the status LED indicator will illuminate **YELLOW.**

Table 10.9. Port Status LED Warning Conditions

Warning Condition	Description
Port power not OK	When the port is active, the monitoring circuitry reports that the output power is not good.
Port input/output over current	When the port is active, the current level measured is higher than the high alarm limit set in its parameter set.
Port input/output under current	When the port is active, the current level measured is lower than the low alarm limit set in its parameter set.
Port input/output over voltage	When the port is active, the current level measured is higher than the high alarm limit set in its parameter set.
Port input/output under voltage	When the port is active, the current level measured is lower than the low alarm limit set in its parameter set.

Table 10.10. Output Power Ports (2 Pin, J5 - J11)

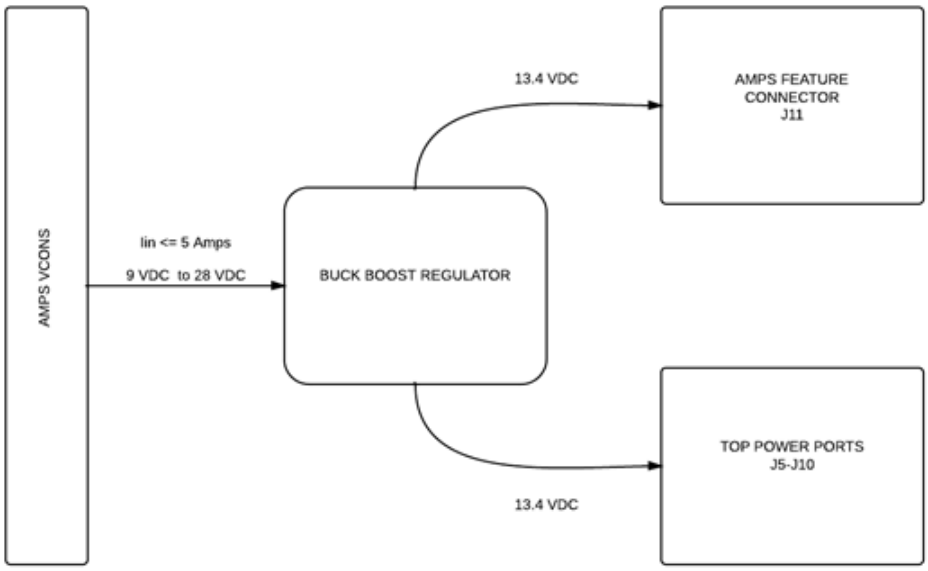
Pin	Signal	Description
1	13.4 VDC	BBR Voltage Output

Liquid Robotics Proprietary and Confidential

Pin	Signal	Description
2	Ground	Return Path

Mating connector: Samtec p/n IPD1-02-S

Figure 10.9. Buck-Boost Power Diagram



Bridge Module

The AMPS Bridge module is a physical interface between two AMPS power domains. It allows the two power domains to work together and to disconnect during failure. There are separate switches for the power buses and the CAN communications bus. Each can be independently controlled. Communications over the CAN bus pass through the bridge module to the CCU.

Module Functions (Commands) - AMPS Bridge modules support the following functions:

- Enable or disable the Bridge power outputs. VCONS and VPROD can be individually controlled, for example, if a power domain that does not use or generate any power on VPROD.
- Enable or disable the CAN bus repeaters.
- Set the soft current limit.

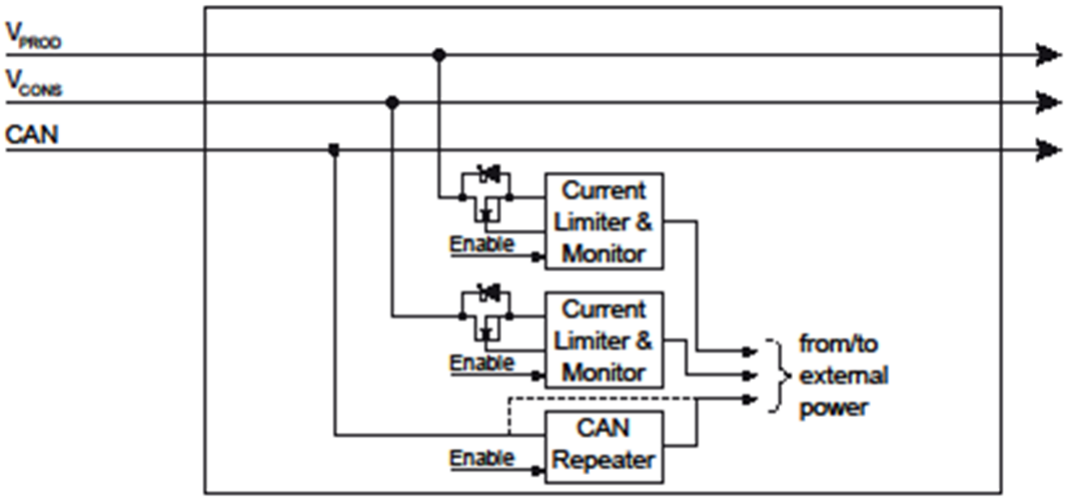
Module Data - AMPS Bridge modules report the following data:

- Energy inbound (negative) or outbound (positive) from VCONS and VPROD. The sum of the energy on both sides of the bridge should not change much over time. Large values of this quantity would represent excessive leakage current.
- The soft current limit.
- The hard current limit (this is not writable, but serves to identify the capability of the hardware). Future bridges may have different current limits.

Liquid Robotics Proprietary and Confidential

Figure 10.10. Bridge Module

Figure 10.11. Bridge Module Electrical Diagram



NOTE: Input bridges bypass the CAN repeater (dotted line).
Output bridges must use the CAN repeater.

1 MPU Module Carrier

The 1 MPU Module Carrier is a passive AMPS backplane designed to house up to 4 AMPS modules and 1 dedicated AMPS Bridge module. The 1 MPU Module Carrier provides connectivity to 1 SV3 Expansion Port and 1 additional AMPS Power Unit. Appropriate Expansion Port Signals are broken on headers to provide the user with convenient access to Expansion Port functionality.

Liquid Robotics Proprietary and Confidential

Figure 10.12. 1 MPU Module Carrier

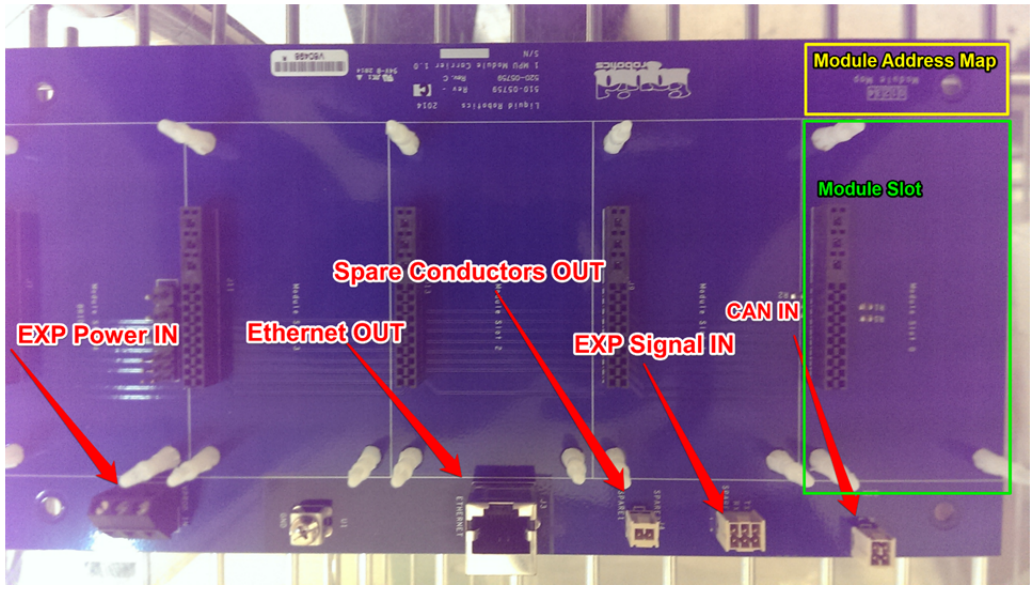


Table 10.11. Electrical Specifications

Parameter	Specification	Notes
Number of AMPS Module Slots	4 + 1 Bridge Slot	5 slots total, 1 slot is dedicated to an AMPS Bridge
AMPS Expansion Port Connectivity	Headers (J2 & J1) to directly mate with an SV3 AMPS Expansion Port	
On board Ethernet (J3)	Standard RJ45 connector to the CCU Expansion Port Ethernet connection	Standard RJ45 connector
Optional Serial Lines, or GPS/ PPS header	Header interfacing with the customizable expansion port spare lines	

Table 10.12. Expansion Port Signals

Pin	Signal	Description
1	Ethernet TX+	10/100 Ethernet TX+
2	Ethernet RX-	10/100 Ethernet RX-
3	Spare1	Configurable Expansion Port Spare Line
4	Ethernet TX-	10/100 Ethernet TX-
5	Ethernet RX+	10/100 Ethernet RX+
6	Spare2	Configurable Expansion Port Spare Line

Liquid Robotics Proprietary and Confidential

Table 10.13. Expansion Port CAN IN

Pin	Signal	Description
1	CANH	
2	CANL	
3, 4	NC	NC

Table 10.14. Expansion Port Power In

Pin	Signal	Description
1	VCONS	Expansion Port VCONS Voltage
2	GND	GND
3	VPROD	Expansion Port VPROD Voltage

Table 10.15. Spare Lines Breakout

Pin	Signal	Description
1	SPARE2	Configurable Expansion Port Spare line
2	SPARE1	Configurable Expansion Port Spare line

Note

The Ethernet Pin-Out is standardized.

3 MPU Module Carrier

The 3 MPU Module Carrier is a passive AMPS backplane designed to house up to 6 AMPS modules and 1 dedicated AMPS Bridge module. The 3 MPU Module Carrier provides connectivity to 1 SV3 Expansion Port and 1 additional AMPS Power Unit. Appropriate Expansion Port Signals are broken on headers to provide the user with convenient access to Expansion Port functionality.

Liquid Robotics Proprietary and Confidential

Figure 10.13. 3 MPU Module Carrier

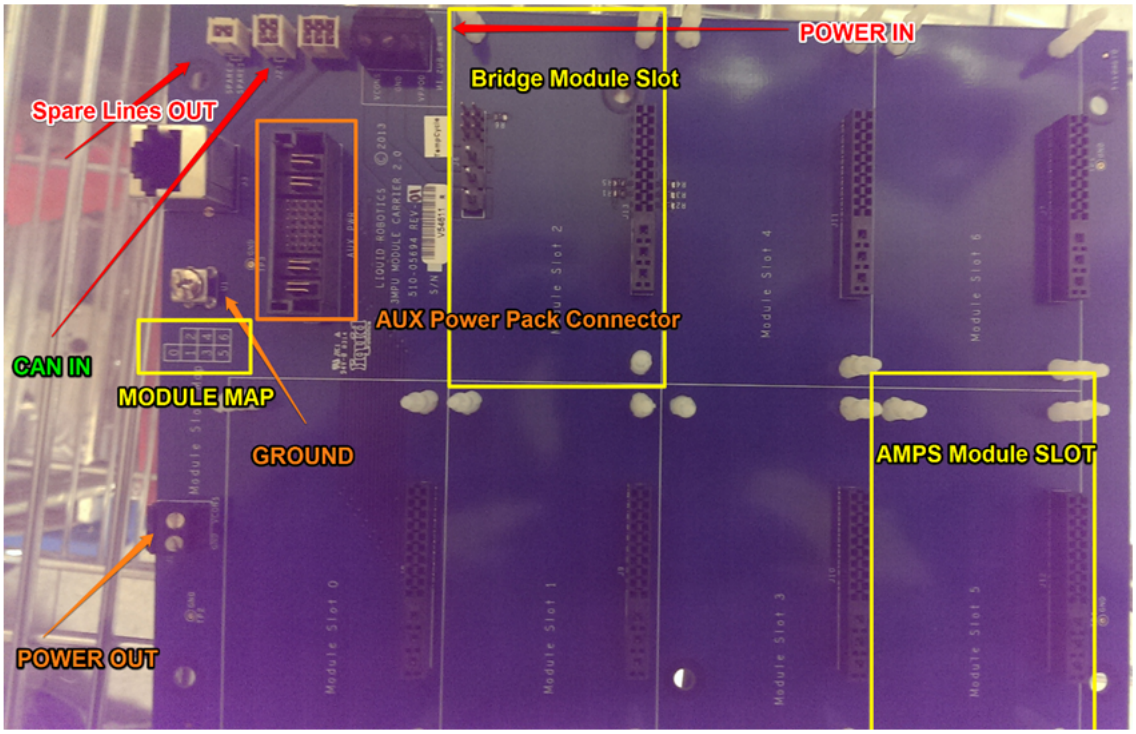


Table 10.16. Electrical Specifications

Parameter	Specification	Notes
Number of AMPS Module Slots	6 + 1 Bridge Slot	7 slots total, 1 slot is dedicated to an AMPS Bridge
AMPS Expansion Port Connectivity	Headers to directly mate with an SV3 AMPS Expansion Port	
Auxiliary Battery Connector	Header to interface with an AMPS Auxiliary Power Pack	
On board Ethernet (J3)	Standard RJ45 connector to the CCU Expansion Port Ethernet connection	Standard RJ45 connector
Optional Serial Lines, or GPS/ PPS header	Header interfacing with the customizable expansion port spare lines	
Expansion Port Power Out	Provided on J22 & Ground Lug U1	

Table 10.17. J2 - Expansion Port Signals

Pin	Signal	Description
1	Ethernet TX+	10/100 Ethernet TX+
2	Ethernet RX-	10/100 Ethernet RX-
3	Spare1	Configurable Expansion Port Spare Line

Liquid Robotics Proprietary and Confidential

Pin	Signal	Description
4	Ethernet TX-	10/100 Ethernet TX-
5	Ethernet RX+	10/100 Ethernet RX+
6	Spare2	Configurable Expansion Port Spare Line

Table 10.18. J21 - Expansion Port CAN IN

Pin	Signal	Description
1	CANH	
2	CANL	
3, 4	NC	NC

Table 10.19. J1 - Expansion Port Power In

Pin	Signal	Description
1	VCONS	Expansion Port VCONS Voltage
2	GND	GND
3	VPROD	Expansion Port VPROD Voltage

Table 10.20. J4 - Spare Lines Breakout

Pin	Signal	Description
1	SPARE2	Configurable Expansion Port Spare line
2	SPARE1	Configurable Expansion Port Spare line

Table 10.21. J22 - Power Breakout

Pin	Signal	Description
1	VCONS	VCONS - Post Bridge
2	GND	Ground

Thrudder PDC

The Power Domain Controller - Thrudder module (PDCT) is the Thrudder power domain's supervisor as well as its command and control relay to the modules residing within the Thrudder domain. It typically communicates with the Vehicle Management Computer (VMC) residing in the CCU using a CAN bus interface that may also be shared with other power domains. The PDCT provides the VMC with the Thrudder power domain status and environmental conditions.

The PDCT is a non-standard AMPS shape, and has some added capabilities, but must be mated to a Thrudder module (Liquid Robotics p/n: 04724 or equivalent). It is designed especially for the Thrudder (rudder-thruster) subsystem.

The PDCT module collects all of the following data for on-board storage and transmission to shore-side operations:

Liquid Robotics Proprietary and Confidential

Dry Box Pressure

Dry Box Temperature

On-board power consumption measurements.

A log of all communications to and from AMPS modules in the power domain that the PDCT controls (for troubleshooting purposes).

A list of all of the modules within the PDCT's including each module's firmware version, board identification information, and operational state.

Functionality of half an AMPS Bridge module, which provides expansion capability to down-line power domains that do not use VPROD. Controls a single CAN bus port and VCONS power bus. Some of its electronics are located on a sister board.

An integrated 3-axis Accelerometer, Gyroscope, Compass. (Not implemented.)

Water detection. Sense wires (traces) located on sister board.

Isolation fault monitoring.

Figure 10.14. AMPS Thrudder Power Domain Controller Module (PDCT)

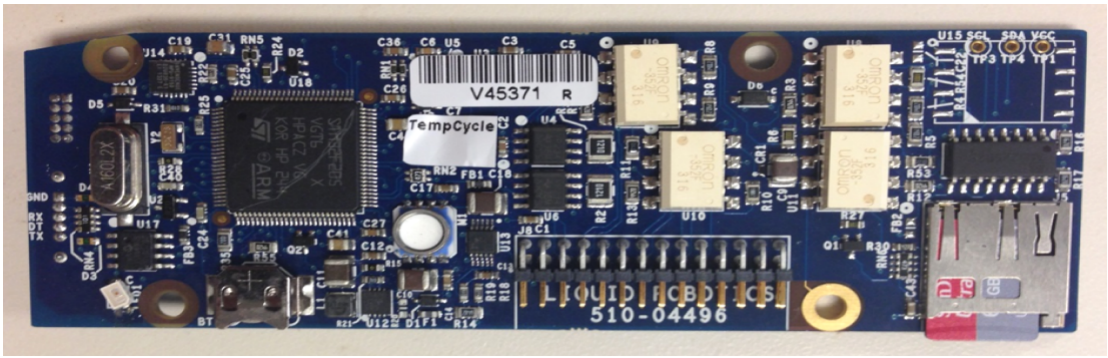


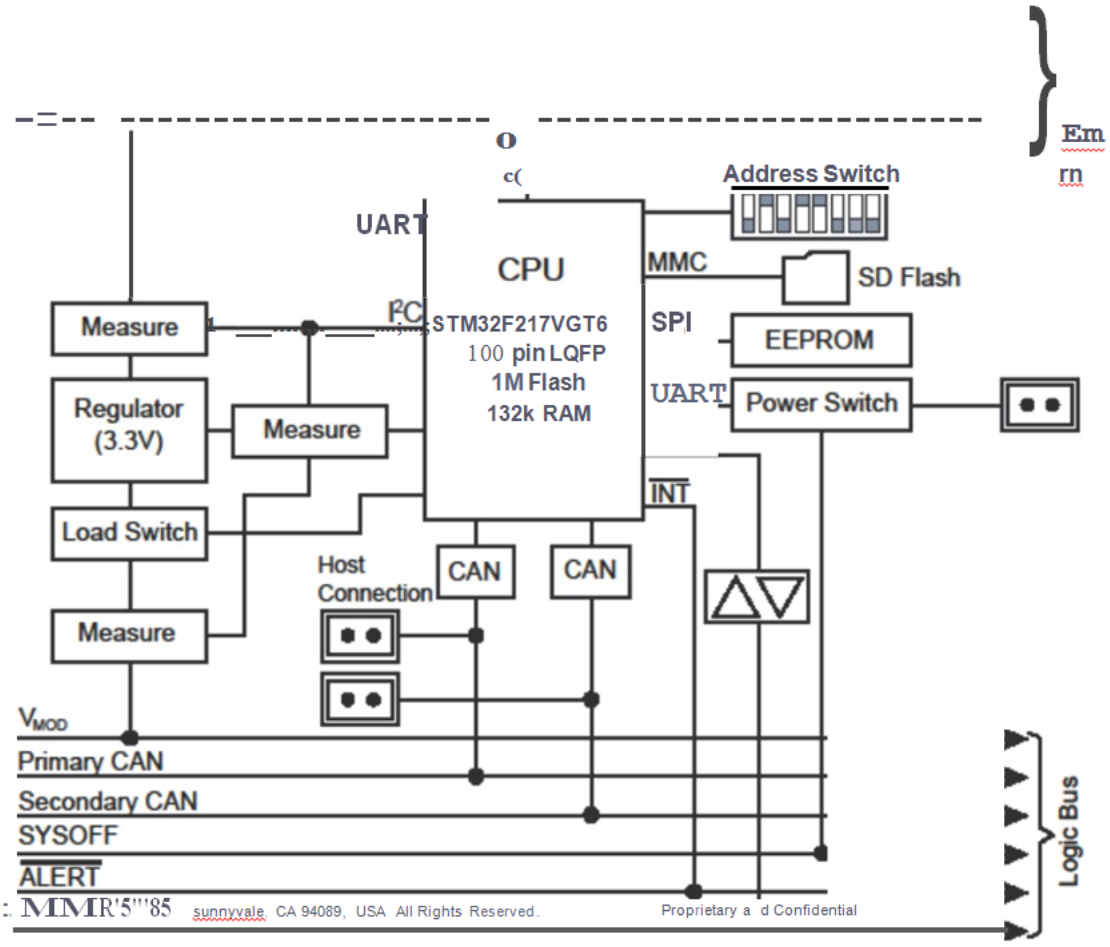
Table 10.22. Console (J9)

Pin	Signal	Description
1	RTS	Ready to Send - not used.
2	RXD	Receive Data (in to board)
3	TXD	Transmit Data (from board)
4	CTS	Clear to Send - not used.
5	GROUND	Signal Ground

This console connector provides a window into the PDC processor. It is a CMOS-level (3.3V) serial port that can be used to observe what is happening within the PDC. **It can also be used to change processor settings - proceed with caution.**

Liquid Robotics Proprietary and Confidential

Figure 10.15. Thrudder Power Domain Controller Electrical Diagram



SMC Carrier

The AMPS SMC Carrier Board is an active back plane. It's primary purpose is to accommodate a Sensor Management Computer (SMC) in an SV3 payload. The SMC Carrier Board can be used in payloads that are connected to an SV3 Expansion Port or an SV3 Sensor Port. The board level shifts the SMC serial lines to RS232 levels and provides break out connectors necessary to wire an SMC into an AMPS based SV3 Expansion Port payload, or a payload connected to an SV3 Sensor Port.

Liquid Robotics Proprietary and Confidential

Figure 10.16. SMC Carrier Board with SMC Mounted

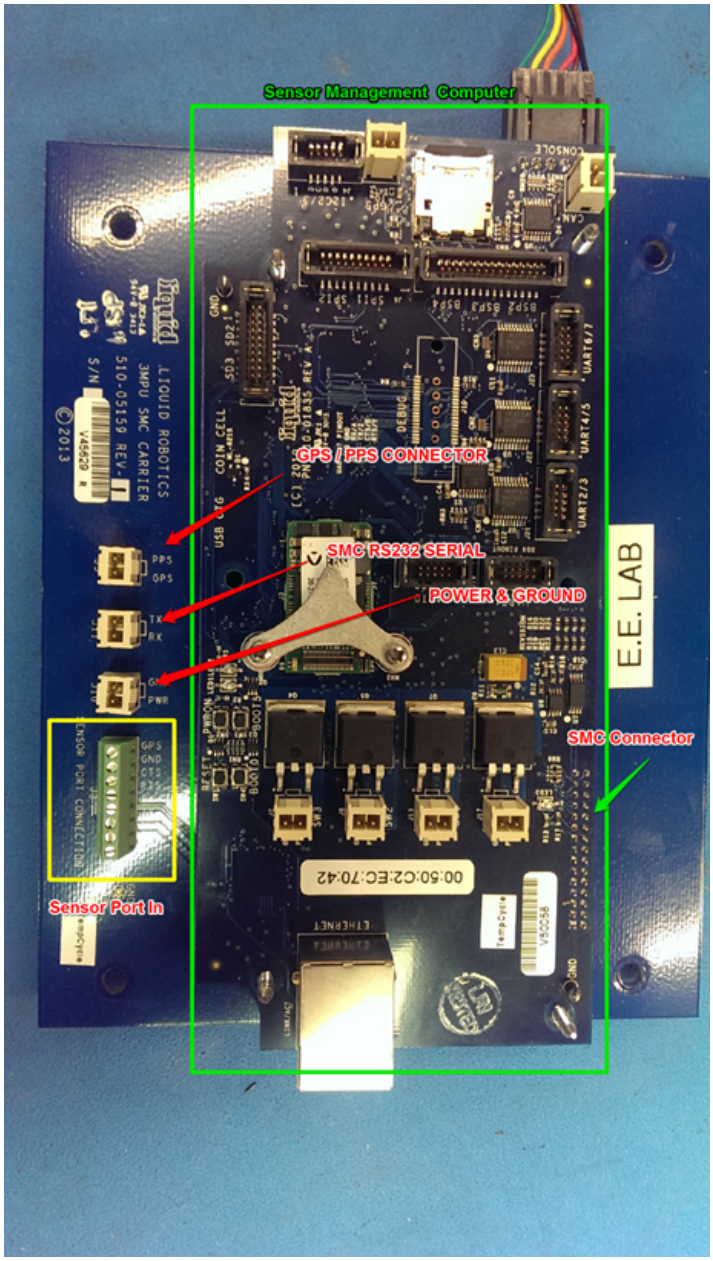


Table 10.23. Electrical Specifications

Parameter	Specification	Notes
Vin	9 - 24 VDC	Higher voltages (>20 VDC) require SMC Part Number 510-01835 Rev. B, or higher.
Serial Lines (RX, TX, RTS, CTS)	RS232 Levels	
GPS/PPS	RS232 Levels	

Liquid Robotics Proprietary and Confidential

Table 10.24. J3 - Sensor Port Landing

Pin	Signal	Description
1	PPS	Pulse Per Second Signal from Sensor Port
2	Vin	Vin
3	TX	RS232 TX from SMC
4	RX	RS232 RX from SMC
5	RTS	Not Terminated
6	CTS	Not Terminated
7	GND	
8	GPS	GPS from Sensor Port

Table 10.25. J5 - GPS/PPS Breakout

Pin	Signal	Description
1	GPS	GPS from Sensor Port
2	PPS	Pulse Per Second Signal from Sensor Port

Table 10.26. J11 - RS232 Breakout

Pin	Signal	Description
1	SMC RX	RS232 RX from the SMC
2	SMC TX	RS232 TX from the SMC

Table 10.27. J10 - Power Breakout

Pin	Signal	Description
1	Vin	Vin
2	GND	Ground

Sensor Management Computer (SMC)

A Sensor Management Computer (SMC) is a single board embedded system running Linux. One or more SMCs can be used for functions the CCU computer is not available to perform. For example, a SMC can be used to:

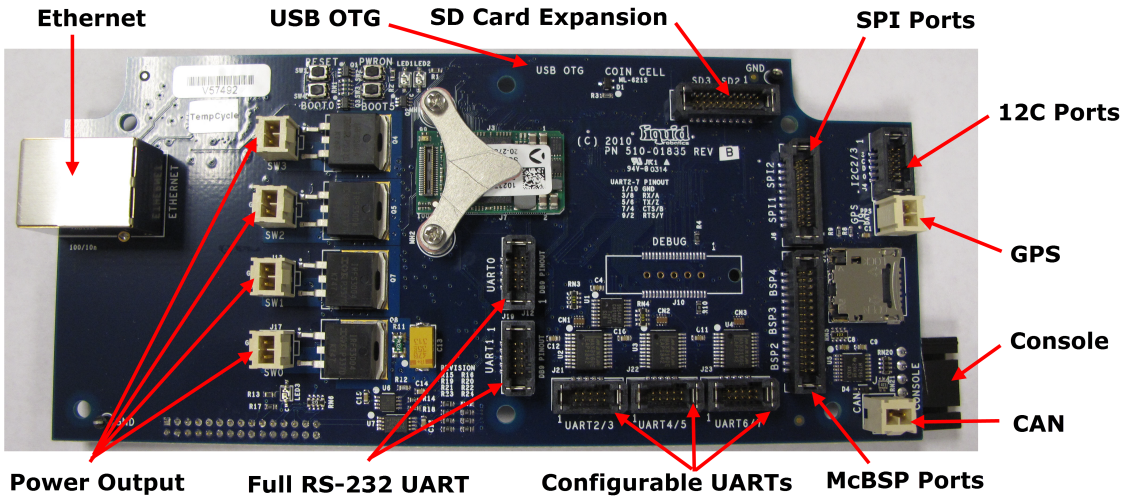
- Host customer or Liquid Robotics developed payload-specific applications
- Run data reduction algorithms
- Run autonomy algorithms

This section provides information on the electrical interfaces to the SMC. For information on developing applications to run on the SMC, see Chapter 11: *Payload Development - Software Environment*.

Overview

This section covers the various connectors and interfaces to the Sensor Management Computer (SMC) part numbers 510-01232 and 510-01835. Each connection is described with a pin-out, part numbers for the mating connector/crimp pins and an electrical specification.

Figure 10.17. SMC Connector Locations



Console Port

The console port is the main interface to the processor module. This port is used to login to the Sensor Management Computer Linux console and is the primary debug point. After deployment, this port is typically unavailable. This connection is intended to connect directly to an FTDI TTL-232R-3V3. The TTL-232R-3V3 should be oriented such that pin 1 (black lead) is to the left when reading CONSOLE.

Console Port Pin Out

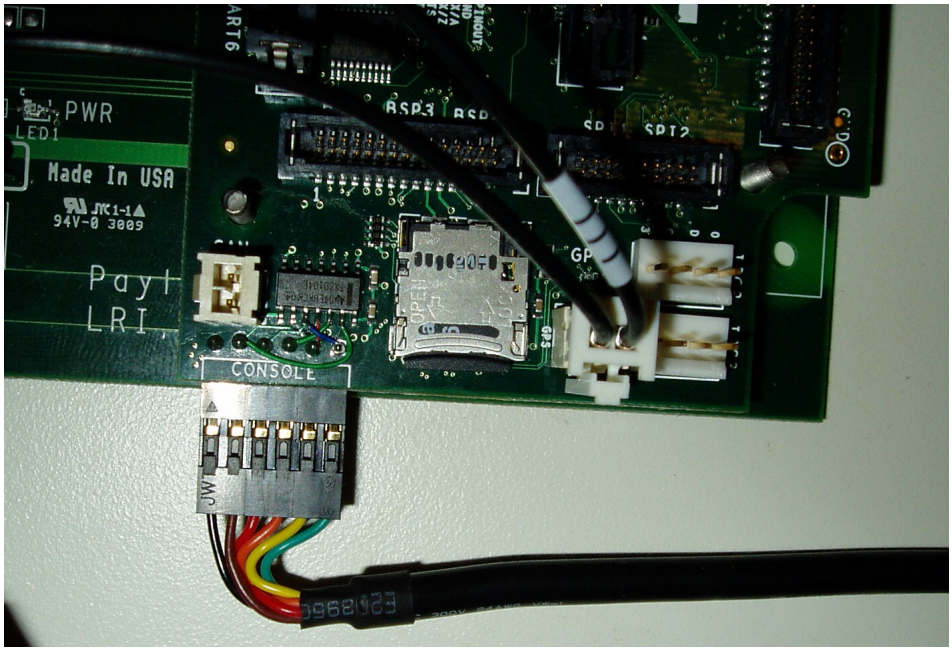
The pin out on the console port is from left to right.

Pin #	SMC Name	Notes
1	GND	System Ground
2	RTS#	Not currently used, but available to support hardware flow control
3	NC	5V signal from USB converter
4	RXD	Data into the SMC
5	TXD	Data out of the SMC
6	CTS#	Not currently used, but available to support hardware flow control

Console Port Electrical Interface

The console port is a 3.3 volt level CMOS serial port intended to be interfaced to a level converter to support either a USB interface or an RS-232 style interface. For a detailed specification of the I/O levels, see the data sheet for the TI TXS0104E.

Figure 10.18. Console Port Connection



Full RS-232 Ports (UART0 and UART1)

The full RS-232 ports implement a complete RS-232 interface that is compatible with a standard DB-9 pin out. One option for connecting up to this port is to use the Liquid Robotics Inc. cable assembly 140-01233 that converts the connector directly to a DB9. The connector is keyed and latching for robustness.

Full RS-232 Port Pin Out

Pin #	SMC Name	Notes
1	CD	Carrier Detect input
2	RX	Data into the UART
3	TX	Data out of the UART
4	DTR	Data Terminal Ready output
5	GND	Ground
6	DSR	Data Set Ready input
7	RTS	Request To Send output
8	CTS	Clear To Send input
9	RI	Ring Indicator input
10	GND	Ground, but intended as a no connect

Full RS-232 Electrical Interface

These connections are compatible with all standard RS-232 drivers.

Liquid Robotics Proprietary and Confidential

Full RS-232 Mating Connector

There are two options for a mating connector from Samtec. The preferred option is to order a pre-built cable assembly such as SFSD-05-28-G-05.00-SR. The length can be adjusted as needed by changing the 4-digit number between the G and SR (i.e. the XX.XX in SFSD-05-28-G-XX.XX-SR). The second option is to use a crimp housing and crimp pins, part numbers ISDF-05-D-M and CC03R/L.

Configurable UART Ports (UART2/3, UART4/5, and UART6/7)

The configurable UART ports implement a dual, software configurable UART port that supports any of the following three options, RS-232 with RTS/CTS support, RS-485 full/half duplex and RS-422. One option for connecting to this port is to use the Liquid Robotics Inc. cable assembly 140-01234 that converts the connector directly to two DB9s. The connector is keyed and latching for robustness.

Configurable UART Port Pin Out

Pin #	SMC Name	Notes
1	GND-A	Ground for even numbered port
2	RTS-B	Request To Send output for odd numbered port
3	RX-A	Data into the UART for the even numbered port
4	CTS-B	Clear To Send input for odd numbered port
5	TX-A	Data out of the UART for the even numbered port
6	TX-B	Data out of the UART for the odd numbered port
7	CTS-A	Clear To Send input for even numbered port
8	RX-B	Data into the UART for the odd numbered port
9	RTS-A	Request To Send output for even numbered port
10	GND-B	Ground for odd numbered port

Configurable UART Electrical Interface

These connections are supported by the Maxim driver IC MAX3160EEAP+

Configurable UART Mating Connector

There are two options for a mating connector from Samtec. The preferred option is to order a pre-built cable assembly such as SFSD-05-28-G-05.00-SR. The length can be adjusted as needed by changing the 4-digit number between the G and SR (i.e. the XX.XX in SFSD-05-28-G-XX.XX-SR). The second option is to use a crimp housing and crimp pins, part numbers ISDF-05-D-M and CC03R/L.

GPS/PPS Port

The GPS/PPS port is the interface to the GPS/PPS data coming in from the Float. This connection connects directly to the incoming connector, to the header on the PIB, or to the provided connector on the SMC carrier board or AMPS module carrier board. The SMC is expecting the GPS string to be at 9600 Baud and contain the following three NMEA strings, GGA, ZDA, and VTG, using RS-232 levels. The connector is keyed and latching for robustness.

Liquid Robotics Proprietary and Confidential

GPS/PPS Port Pin Out

Pin #	SMC Name	Notes
1	GPS	GPS Data into the system – Connects to pin 8 of the PIB header
2	PPS	PPS timing pulse into the system – connects to pin 1 of the PIB header

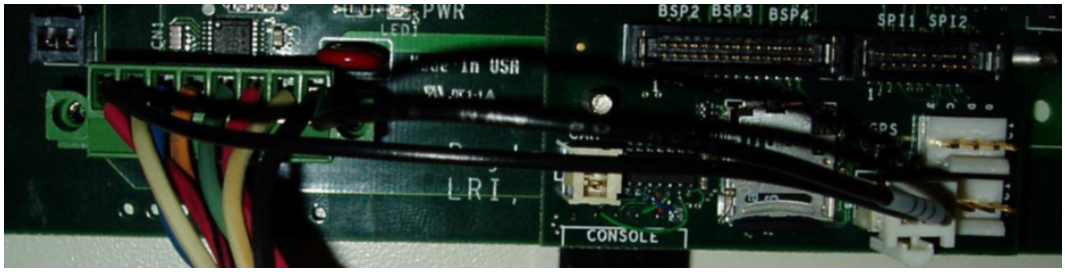
GPS/PPS Electrical Interface

These inputs are capable of handling +/-15 volt signals. They are intended to level shift RS-232 signals down to 1.8 volt CMOS logic. These inputs are simple N-FET level shifts/inverters.

GPS/PPS Mating Connector

There are two options for a mating connector from Samtec. The preferred option is to order a pre-built cable assembly such as MMSS-02-24-L-05.00-S-K. The length can be adjusted as needed by changing the 4-digit number between the L and S (i.e. the XX.XX in MMSS-02-24-L-XX.XX-S-K). The second option is to use a crimp housing and crimp pins, part numbers IPD1-02-S-K and CC79R/L.

Figure 10.19. GPS/PPS Connections to a PIB



Power Ports

The power ports are software controllable switches to enable/disable sensors as needed. The connector is keyed and latching for robustness.

Power Port Pin Out

Pin #	SMC Name	Notes
1	PWR	12 Volt +/-10% output at up to 2 A
2	GND	Ground

Power Electrical Interface

These outputs are capable of providing 12 volts +/-10% at up to 2 amps. The total system current only supports up to 3 amps into the SMAC so the sum of currents out is the limiting factor, not the switches. The current through the switch can be read in software.

Power Mating Connector

There are two options for a mating connector from Samtec. The preferred option is to order a pre-built cable assembly such as MMSS-02-24-L-05.00-S-K. The length can be adjusted as needed by changing

Liquid Robotics Proprietary and Confidential

the 4-digit number between the L and S (i.e. the XX.XX in MMSS-02-24-L-XX.XX-S-K). The second option is to use a crimp housing and crimp pins, part numbers IPD1-02-S-K and CC79R/L.

CAN Port

The CAN port is not currently used or supported, but is intended for inter-SMC communication and/or for specific sensors that require a CAN interface. The information provided here is for future use only. The connector is keyed and latching for robustness.

CAN Port Pin Out

Pin #	SMC Name	Notes
1	CANH	CAN non-inverting output
2	CANL	CAN inverting output

CAN Electrical Interface

This output is supported by a TI SN65HVD230DR transceiver. One thing to note is that this interface is running at 3.3 volts, and although this meets the ISO 11898 standard (CAN specifications), it may not be compatible with older 5.0 volt CAN bus systems.

CAN Mating Connector

There are two options for a mating connector from Samtec. The recommended option is to order a pre-built cable assembly such as MMSS-02-24-L-05.00-S-K. The length can be adjusted as needed by changing the 4-digit number between the L and S (i.e. the XX.XX in MMSS-02-24-L-XX.XX-S-K). The second option is to use a crimp housing and crimp pins, part numbers IPD1-02-S-K and CC79R/L.

USB Ports (not USB OTG)

These ports are not supported.

USB OTG Port

The USB OTG port meets the USB OTG specification. The USB OTG interface is currently not supported in the Linux kernel. To use the USB OTG port as the host a special mini A to B (or mini B) cable is required. A standard A to mini B cable will turn the SMC into a slave device.

USB OTG Port Pin Out

See the USB OTG specification.

USB OTG Electrical Interface

See the USB OTG specification.

USB OTG Mating Connector

See the USB OTG specification.

Liquid Robotics Proprietary and Confidential

Ethernet Port

The Ethernet port meets the 10/100 Base T Ethernet specification. The connector is specified as a latching RJ-45.

Ethernet Port Pin Out

See the Ethernet specification.

Ethernet Electrical Interface

See the Ethernet specification.

Ethernet Mating Connector

See the Ethernet specification.

SD Card Expansion Port

The SD Card Expansion port supports additional data storage. This interface connects directly to the Torpedo module/OMAP processor so the signals must be level shifted to 1.8 volts. This port is untested.

SD Card Expansion Port Pin Out

Pin #	SMC Name	Notes
1	1V8	1.8 Volt output for supplying a logic level reference, limit draw to less than 10mA
2	3V3	3.3 volts for powering any needed circuits
3	SD2_CLK	SD card interface #2 clock output
4	SD2_CDn	SD card interface #2 card detect input
5	SD2_DAT3	SD card interface #2 data bit 3 I/O
6	SD2_CMD	SD card interface #2 command output
7	SD2_DAT1	SD card interface #2 data bit 1 I/O
8	SD2_DAT2	SD card interface #2 data bit 2 I/O
9	GND	Ground
10	SD2_DAT0	SD card interface #2 data bit 0 I/O
11	1V8	1.8 Volt output for supplying a logic level reference, limit draw to less than 10mA
12	3V3	3.3 volts for powering any needed circuits
13	SD3_CLK	SD card interface #3 clock output
14	SD3_CDn	SD card interface #3 card detect input
15	SD3_DAT3	SD card interface #3 data bit 3 I/O
16	SD3_CMD	SD card interface #3 command output
17	SD3_DAT1	SD card interface #3 data bit 1 I/O
18	SD3_DAT2	SD card interface #3 data bit 2 I/O

Liquid Robotics Proprietary and Confidential

Pin #	SMC Name	Notes
19	GND	Ground
20	SD3_DAT0	SD card interface #3 data bit 0 I/O

SD Card Expansion Electrical Interface

The inputs and outputs here are referenced to the OMAP 1.8 volt logic levels. The 1.8 volt output is intended only for level shifters as needed and the rest of the circuit should be run from 3.3 volts.

SD Card Expansion Mating Connector

There are two options for a mating connector from Samtec. The recommended option is to order a pre-built cable assembly such as SFSD-10-28-G-05.00-SR. The length can be adjusted as needed by changing the 4-digit number between the G and SR (i.e. the XX.XX in SFSD-10-28-G-XX.XX-SR). The second option is to use a crimp housing and crimp pins, part numbers ISDF-10-D-M and CC03R/L.

SPI Expansion Port

The SPI Expansion port supports additional circuitry for expanded operation. This interface connects directly to the Torpedo module/OMAP processor so the signals will need to be level shifted to 1.8 volts. This port is untested.

SPI Expansion Port Pin Out

Pin #	SMC Name	Notes
1	1V8	1.8 Volt output for supplying a logic level reference, limit draw to less than 10mA
2	3V3	3.3 volts for powering any needed circuits
3	SPI1_CS2	SPI interface #1 chip select 2 output
4	SPI1_CS3	SPI interface #1 chip select 3 output
5	SPI1_CS0	SPI interface #1 chip select 0 output
6	SPI1_CS1	SPI interface #1 chip select 1 output
7	SPI1_SIMO	SPI interface #1 slave in master out output
8	SPI1_CLK	SPI interface #1 clock output
9	GND	Ground
10	SPI1_SOMI	SPI interface #1 slave out master in input
11	1V8	1.8 Volt output for supplying a logic level reference, limit draw to less than 10mA
12	3V3	3.3 volts for powering any needed circuits
13	SPI2_CS2	SPI interface #2 chip select 2 output
14	SPI2_CS3	SPI interface #2 chip select 3 output
15	SPI2_CS0	SPI interface #2 chip select 0 output
16	SPI2_CS1	SPI interface #2 chip select 1 output
17	SPI2_SIMO	SPI interface #2 slave in master out output
18	SPI2_CLK	SPI interface #2 clock output

Liquid Robotics Proprietary and Confidential

Pin #	SMC Name	Notes
19	GND	Ground
20	SPI2_SOMI	SPI interface #2 slave out master in input

SPI Expansion Electrical Interface

The inputs and outputs here are referenced the OMAP 1.8 volt logic levels. The 1.8 volt output is intended only for level shifters as needed and the rest of the circuit should be run from 3.3 volts.

SPI Expansion Mating Connector

There are two options for a mating connector from Samtec. The recommended option is to order a pre-built cable assembly such as SFSD-10-28-G-05.00-SR. The length can be adjusted as needed by changing the 4-digit number between the G and SR (i.e. the XX.XX in SFSD-10-28-G-XX.XX-SR). The second option is to use a crimp housing and crimp pins, part numbers ISDF-10-D-M and CC03R/L.

McBSP Expansion Port

The McBSP Expansion port supports additional circuitry for expanded operation. This interface connects directly to the Torpedo module/OMAP processor so the signals will need to be level shifted to 1.8 volts. This port is untested.

McBSP Expansion Port Pin Out

Pin #	SMC Name	Notes
1	1V8	1.8 Volt output for supplying a logic level reference, limit draw to less than 10mA
2	3V3	3.3 volts for powering any needed circuits
3	BSP2_I/O1	McBSP interface #2 general purpose I/O 1
4	BSP2_I/O0	McBSP interface #2 general purpose I/O 0
5	BSP2_DR	McBSP interface #2 DR
6	BSP2_I/O2	McBSP interface #2 general purpose I/O 2
7	BSP2_FSX	McBSP interface #2 FSX
8	BSP2_DX	McBSP interface #2 DX
9	GND	Ground
10	BSP2_CLKX	McBSP interface #2 CLKX
11	1V8	1.8 Volt output for supplying a logic level reference, limit draw to less than 10mA
12	3V3	3.3 volts for powering any needed circuits
13	BSP3_I/O1	McBSP interface #3 general purpose I/O 1
14	BSP3_I/O0	McBSP interface #3 general purpose I/O 0
15	BSP3_DR	McBSP interface #3 DR
16	BSP3_I/O2	McBSP interface #3 general purpose I/O 2
17	BSP3_FSX	McBSP interface #3 FSX
18	BSP3_DX	McBSP interface #3 DX

Liquid Robotics Proprietary and Confidential

Pin #	SMC Name	Notes
19	GND	Ground
20	BSP3_CLKX	McBSP interface #3 CLKX
21	1V8	1.8 Volt output for supplying a logic level reference, limit draw to less than 10mA
22	3V3	3.3 volts for powering any needed circuits
23	BSP4_I/O1	McBSP interface #4 general purpose I/O 1
24	BSP4_I/O0	McBSP interface #4 general purpose I/O 0
25	BSP4_DR	McBSP interface #4 DR
26	BSP4_I/O2	McBSP interface #4 general purpose I/O 2
27	BSP4_FSX	McBSP interface #4 FSX
28	BSP4_DX	McBSP interface #4 DX
29	GND	Ground
30	BSP4_CLKX	McBSP interface #4 CLKX

McBSP Expansion Electrical Interface

The inputs and outputs here are referenced to the OMAP 1.8 volt logic levels. The 1.8 volt output is intended only for level shifters as needed and the rest of the circuit should be run from 3.3 volts.

McBSP Expansion Mating Connector

There are two options for a mating connector from Samtec. The recommended option is to order a pre-built cable assembly such as SFSD-15-28-G-05.00-SR. The length can be adjusted as needed by changing the 4-digit number between the G and SR (i.e. the XX.XX in SFSD-15-28-G-XX.XX-SR). The second option is to use a crimp housing and crimp pins, part numbers ISDF-15-D-M and CC03R/L.I2C Ports (I2C2 and I2C3)

I2C Ports

The I2C ports are for debug only in the 510-00973 design. In the 510-01232 design, they have an expansion capability as specified below. The 510-00973 connectors are keyed. The 510-01232 PCBA connector is keyed and latching for robustness.

I2C Ports 510-01232 Pin Out

Pin #	SMC Name	Notes
1	3V3	3.3 volt output to power external circuits
2	3V3	3.3 volt output to power external circuits
3	I2C2_SCL	I2C2 serial clock output
4	I2C3_SCL	I2C3 serial clock output
5	I2C2_SDA	I2C2 serial data I/O
6	I2C3_SDA	I2C3 serial data I/O
7	GND	Ground
8	GND	Ground

Liquid Robotics Proprietary and Confidential

Pin #	SMC Name	Notes
9	GND	Ground
10	GND	Ground

I2C Electrical Interface

This interface conforms to the I2C specification for 3.3 volt logic levels and supports up to 400 KHz operation.

I2C Ports 510-01232 Mating Connector

There are two options for a mating connector from Samtec. The recommended option is to order a pre-built cable assembly such as SFSD-05-28-G-05.00-SR. The length can be adjusted as needed by changing the 4-digit number between the G and SR (i.e. the XX.XX in SFSD-05-28-G-XX.XX-SR). The second option is to use a crimp housing and crimp pins, part numbers ISDF-05-D-M and CC03R/L.

Chapter 11. Payload Development - Software Environment

This chapter provides information on the following payload development topics:

- "Regulus" (page 228)
- "SV3 Manifests" (page 228)
- "How Remote Applications Handle Manifest Files" (page 229)
- "Parsing the Manifests" (page 230)
- "Expansion Port Payload Registration with Regulus" (page 239)
- "Regulus Expansion Port REST Interface (XPREST)" (page 243)

.

Regulus

Regulus is a Wave Glider SV3 management application that runs on Linux on the Vehicle Management Computer (VMC) in the Command and Control Unit (CCU). Regulus controls all Wave Glider SV3 operations, including navigation and management of telemetry and data.

Regulus also manages Wave Glider SV3 communications for most payloads, with the exception of payloads that have been developed to provide exclusive independent communications channels.

A Regulus Stub that includes the Iridium Relay and Virtual Relay modules runs at the remote end of the communications channels (on ship or onshore). The Regulus Stub manages security keys and communications channels. Regulus can be configured to automatically select the lowest cost available communications channel for each data and command transmission.

SV3 Manifests

Regulus generates two JSON files referred to as manifests. The manifests contain extensive information on the Vehicle.

The Configuration Manifest provides a detailed accounting of the Vehicle configuration including base Wave Glider components, Liquid Robotics factory options, and integrated payload devices that are registered with Regulus and connected to CCU Expansion Ports. For information on CCU Expansion Ports, see Chapter 3: *Electrical Specifications*. For information on registering payload devices with Regulus see "Expansion Port Payload Registration with Regulus" (page 239).

The Commands Manifest lists the commands that are supported by the Vehicle, including commands specific to payload devices that are registered with Regulus and connected to CCU Expansion Ports.

Both manifests can be retrieved on command manually, or automatically, by remote applications. For example, when Regulus notifies WGMS that new manifests have been published, WGMS can

Liquid Robotics Proprietary and Confidential

automatically retrieve the manifests based on criteria contained in the Regulus notification that indicates a significant (non-trivial) change has occurred.

The manifest files are transmitted compressed, and then decompressed to a JSON format by the remote application. For more information see "How Remote Applications Handle Manifest Files" (page 229).

The two manifests can be parsed by remote applications (for example, WGMS) to populate a UI with vehicle specific commands and data. For information on parsing the manifests to populate a remote application UI see "Parsing the Manifests" (page 230).

To view a Configuration Manifest see Appendix B: *Configuration Manifest Example*

To view a Commands Manifest see Appendix C: *Commands Manifest Example*.

How Remote Applications Handle Manifest Files

The file name for each manifest file identifies the Vehicle that generated it and includes a time stamp indicating when it was generated.

The manifest file names include the Vehicle unique id:

The file name format for the compressed Configuration Manifest is:

`<hexadecimal vehicle ID>.GZP`,

which decompresses to a JSON file named: `<hexadecimal vehicle ID>.<application specific extension>` For example, with WGMS the JSON file name used is: `<hexadecimal vehicle ID>.TXP`

The file name format for the compressed Commands Manifest is:

`<hexadecimal vehicle ID>.GZC`,

which decompresses to a JSON file named: `<hexadecimal vehicle ID>.<application specific extension>` For example, with WGMS the JSON file name used is: `<hexadecimal vehicle ID>.TXC`

The following order of events is typical of how a remote application handles manifest files:

1. The Vehicle (Regulus) transmits a compressed manifest file to the remote application, for example, to WGMS.

   The Vehicle (Regulus) sends a **New File Available** message to the remote application (WGMS) containing the file name, the size of the file, and the checksum of the file. For information on the **New File Available** message see "New File Available (0x60)" (page 103).

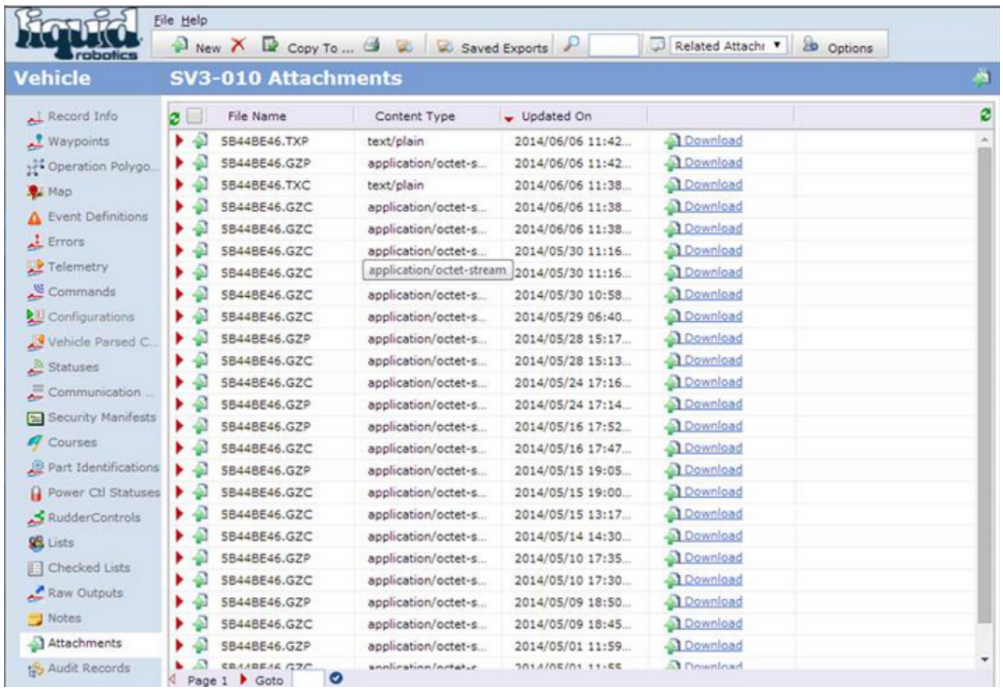
2. The remote application (WGMS) compares the file size and checksum to those of the most recent manifest file of the same type (Configuration Manifest or Command Manifest) that it received from the Vehicle (Regulus), to determine if it needs to transfer the file (based on programmable criteria).

3. If the remote application (WGMS) needs the manifest file, it issues a **Read File Request** command to the Vehicle (Regulus). For information on the **Read File Request** command see "Read File Request (0x9003)" (page 275).

Liquid Robotics Proprietary and Confidential

4. When the compressed manifest file is received by the remote application (WGMS), it is decompressed using GZIP and stored in the JSON format. For example, WGMS stores decompressed manifest files as attachments to the Vehicle as illustrated in Figure 11.1.

Figure 11.1. Manifest Files as WGMS Attachments



Parsing the Manifests

The following steps provide an overview of the manifest parsing process:

1. The Vehicle (Regulus) transmits compressed manifests (.GZC and .GZP files) to the remote application.

2. The back end of the remote application uses GZIP to decompress the manifest files:

   - The compressed Configuration Manifest .GZP file is converted to a JSON file with an application specific file extension (for example, WGMS uses .TXP).

   - The compressed Commands Manifest .GZC file is converted to a JSON file with an application specific file extension (for example, WGMS uses .TXC).

3. The front end of the remote application reads the decompressed manifest files and populates command button names, tool tips, field labels and so forth in a UI template, based on specific parameters in the two manifest files.

Each Wave Glider SV3 Vehicle generates manifest files that reflect the configuration and commands that are registered with Regulus on that Vehicle. The remote application UI (for example the WGMS SV3 Commands Window) for one Vehicle may not contain exactly the same Services, Devices, and Payload commands as the UI for another Vehicle. In this way the only commands that appear in the UI for any SV3 Vehicle are the commands associated with the Devices, Payloads, and Services that are available on that Vehicle.

Liquid Robotics Proprietary and Confidential

If you are integrating a payload on the Wave Glider SV3 platform and you are using WGMS as a remote application, you do not need to write code to parse the manifests. WGMS will handle this for you, provided the manifests are properly formatted. For information on how to generate properly formatted manifests see "Expansion Port Payload Registration with Regulus" (page 239) and "Regulus Expansion Port REST Interface (XPREST)" (page 243).

If you plan to use a remote application other than WGMS, you will still need to generate properly formatted manifests as described in "Expansion Port Payload Registration with Regulus" (page 239) and "Regulus Expansion Port REST Interface (XPREST)" (page 243), but you will also need to develop code that parses the manifests to populate command button names, tool tips, field labels and so forth in the UI template that you use. The following sections provide information you will need when developing code to parse the manifest files.

Configuration Manifest Organization

This section focuses on the organization of the Configuration Manifest (.TXP) file and on the Configuration Manifest parameters that can be parsed to populate a remote application UI template.

The following manifest snippet is provided to illustrate how the Configuration Manifest is organized.

Note

To view a complete Configuration Manifest see Appendix B: *Configuration Manifest Example*

Configuration Manifest Snippet

```
{
    "amps": {
        "domains": {
            "host": {
                "manifest": {
                    "Description": "Host Power Domain",
                    "DomainIndex": 1,
                    "DomainName": "Host",
                    "DomainSerial": "0x3332470b3236303200240022",
                    "ParentID": ""
                },
                "wiring": {
                    "Connectors": {
                        "G1": {
                            "module": 11,
                            "port": 256
                        },
                        "G2": {
                            "module": 12,
                            "port": 256
                        },
                        }
                    }
                }
            },
            "thrudder": {
                "manifest": {
                    "Description": "Thrudder Power Domain",
                    "DomainIndex": 2,
                    "DomainName": "",
```

Liquid Robotics Proprietary and Confidential


```
                "DomainSerial": "0x00000000",
                "ParentID": ""
            },
            "wiring": {
                "Connectors": {
                    "ICRudder": {
                        "module": 1,
                        "port": 2
                    },
                    }
                }
            }
        }
    },
"devices": {
    "expansionPort1": {
        "boundAmpsDomain": "",
        "class": "com.liquidr.regulus.SV3payloads.SV3PayloadDevice",
        "enable": true,
        "generateAlarms": false,
        "ip": "192.168.103.0",
        "name": "xp1",
        "on": false,
        "payloadPortName": "G1",
        "postPowerOff": "ifdown eth0.103",
        "postPowerOn": "sleep 30;ifup eth0.103",
        "reportAmpsErrors": false,
        "reportAmpsWarnings": false,
        "smcInitialPower": true,
        "wiring": {
            "connector": "G1",
            "domain": "host"
        }
    },
    "lightBar": {
        "class": "com.liquidr.regulus.devices.PoweredDevice",
        "enable": true,
        "generateAlarms": false,
        "on": false,
        "reportAmpsErrors": false,
        "reportAmpsWarnings": false,
        "wiring": {
            "connector": "O2Light",
            "domain": "host"
        }

},
"id": {
    "avatar": "http://liquidr.github.io/avatars/DefaultAvatar.png",
    "imei": "300434060722240",
    "lavatar": "",
    "longname": "SV3-030",
    "requires": "liquidr",
    "shortname": "SV3-030",
    "sn": 141824934
},
"logfile": "~/Regulus/log/run-%d.log",
"payload": {
    "xp0": {},
```

Liquid Robotics Proprietary and Confidential


```
        "xp1": {},
        "xp2": {},
        "xp3": {},
        "xpSub": {}
    },
    "services": {
        "AIS": {
            "enable": true,
            "generateAlarms": false,
            "logHitDistances": false,
            "logPSHI": false
        },
        "navigator": {
            "achieveThreshholdMeters": 25,
            "avoidobstacles": true,
            "boostoff": 1.1,
            "booston": 0,
            "enableLineFollowing": true,
            "generateAlarms": false,
            "lineFollowingStrengthObj": 1,
            "lineFollowingWidth": 25,
            "lookAhead": 5,
            "rudderForceLimit": 5
        }
    }
}
```

Each Configuration Manifest begins with information about the AMPs domains on the Vehicle. In the Commands Manifest Snippet above, there are two `amps` domains. One `domain` is named `host` and the other `domain` is named `thrudder`. Following each `domain` name are parameters that define the `domain` and detail the `wiring` and `Connectors` associated with that domain. Each Vehicle will have at least one `Host Power Domain` and a single `Thrudder Power Domain`. For more information on the AMPS and Thrudder electrical connections see Chapter 3: *Electrical Specifications*. Following the `amps` domains there are three other categories of Configuration Manifest entries:

- `devices` - These are standard and optional Liquid Robotics devices that are included in the Vehicle configuration. When WGMS is used as the remote application, these entries appear in the **Devices** menu of the SV3 Commands window.
- `payload` - These include payloads provided by Liquid Robotics and payloads developed by integrators. When WGMS is used as the remote application, these entries appear in the **Payloads** menu of the SV3 Commands window.
- `services` - These are services that are available in the Vehicle configuration. When WGMS is used as the remote application, these entries appear in the **Services** menu of the SV3 Commands window.

At the same hierarchical level as `devices`, `payload`, and `services`, each Configuration Manifest also has an `id` category and a `logfile` category. The `id` category lists the unique identification information for the Vehicle, and the `logfile` category contains the file path for the Vehicle logs on Regulus.

Table 11.1 lists all of the Configuration Manifest parameters with a description of each parameter.

Liquid Robotics Proprietary and Confidential

Table 11.1. Configuration Manifest Parameters

Parameter	Description
`amps`	Contains information on the Adaptable Modular Power System (AMPS) configuration on the Vehicle, which can have multiple power domains. For more information on the AMPS see Chapter 3: *Electrical Specifications*.
`domains`	Contains information on each AMPS power domain in the AMPS configuration on the Vehicle.
<domain name>	The name of an AMPS power domain listed under `domains`. In the Configuration Manifest Snippet, the two AMPS domain names listed are `host` and `thrudder`.
`manifest`	Contains the parameters that define a specific AMPS power domain.
`Description`	The description of an AMPS power domain.
`DomainIndex`	The index of an AMPS power domain.
`DomainName`	The name of an AMPS power domain - may not be identical to <domain name>.
`DomainSerial`	
`ParentID`	
`wiring`	Contains wiring information for the AMPS domain.
`Connectors`	Contains connector information for the AMPS domain.
<connector name>	The name of each connector. In the Configuration Manifest Snippet, the two connector names in the `host` domain are `G1` and `G2`. In the `thrudder` domain there is one connector named `ICRudder`.
`module`	
`port`	
`devices`	Contains information on each device included in the Vehicle configuration.
<device name>	This is the name of a device, for example, `expansionPort1`, and `lightBar`.
`boundAmpsDomain`	
`class`	
`enable`	
`generateAlarms`	
`ip`	
`name`	
`on`	
`payloadPortName`	
`postPowerOff`	
postPowerOn	
`reportAmpsErrors`	
`reportAmpsWarnings`	
`smcInitialPower`	

Liquid Robotics Proprietary and Confidential

Parameter	Description
`wiring`	Contains wiring information for the device.
`connector`	The name of the CCU connector the device is connected to.
`domain`	The name of the AMPS domain the device is included in.
`id`	Lists the unique identification information for the Vehicle.
`logfile`	Contains the file path for the Vehicle logs on Regulus.
`payload`	Contains information on each payload included in the Vehicle configuration.
`xp0`	Reserved to contain information on payloads that cannot be identified using the IP address, or are resident on the VMC external to Regulus.
`xp1`	Contains information on any payloads connected to the expansion port 1 (G1).
`xp2`	Contains information on any payloads connected to the expansion port 2 (G2).
`xp3`	Contains information on any payloads connected to the expansion port 3 (G3).
`xpSub`	Contains information on any payloads connected to the sub expansion port.
`services`	Contains information on each service available the Vehicle configuration.
<service name>	The name a service that is available in the Vehicle configuration. In the Configuration Manifest Snippet, the two service names listed are `AIS` and `navigator`.

Commands Manifest Organization

This section focuses on the organization of the Commands Manifest (.TXC) file and on the Commands Manifest parameters that can be parsed to populate a remote application UI template. The following manifest snippet is provided to illustrate how the Commands Manifest is organized.

Note

To view a complete Commands Manifest see Appendix C: *Commands Manifest Example*.

Commands Manifest Snippet

```
{
    "id": {
        "avatar": "http://liquidr.github.io/avatars/DefaultAvatar.png",
        "imei": "300434060722240",
        "lavatar": "",
        "longname": "SV3-030",
        "requires": "liquidr",
        "shortname": "SV3-030",
        "sn": 141824934
    },
    "usedByPilots": [
        {
            "label": "Avoid Obstacles",
            "method": "setAvoidObstacles",
            "paramLabels": [
                "Select 'True' to turn on obstacle avoidance"
```

Liquid Robotics Proprietary and Confidential


```
            ],
            "paramTypes": [
                "boolean"
            ],
            "tag": "navigator",
            "tooltip": "Automatic avoidance of dynamic obstacles - primarily ships.",
            "var": "ffn"
        },
        {
            "expert": true,
            "label": "Force Rudder",
            "method": "forceRudderProtected",
            "paramLabels": [
                "Forced rudder angle in degrees"
            ],
            "paramTypes": [
                "float"
            ],
            "tag": "navigator",
            "tooltip": "Suspend all navigation settings and force the rudder to a fixed angle.  Use
             'Resume' to restore navigation settings.", "var": "ffn"
        },
    ],
    "varNames": {
        "AISReporter": "com.liquidr.regulus.wgmsinterop.AISReporter",
        "CPUTempMonitor": "com.liquidr.regulus.services.CPUTempMonitor",
        "Distance2Force": "com.liquidr.regulus.nav.Distance2Force$Delegate",
        "Eval": "com.liquidr.regulus.eval.Eval",
        "Forcefield": "com.liquidr.regulus.nav.Forcefield$Watch8Attractor",

    }
}
```

Table 11.2 lists all of the Command Manifest parameters with a description of each parameter.

Table 11.2. Command Manifest Parameters

Parameter	Description
`id`	Contains information about the Vehicle.
`avatar`	A unique (among Wave Gliders) icon that identifies the Vehicle.
`imei`	The IMEI number associated with the modem on the Vehicle.
`lavatar`	
`longname`	
`requires`	
`shortname`	
`sn`	The Vehicle serial number.
`usedByPilots`	Contains commands that are commonly used by Pilots. For example, when the Commands Manifest is parsed for WGMS SV3 Commands UI, this parameter allows a command view that is Piloting-centric, as opposed to displaying all of the commands supported by the Vehicle configuration.
`expert`	`"expert": true` appears on the line preceding the `label` parameter for expert level commands in the Commands Manifest. For example, when the Commands Manifest is parsed for the WGMS SV3 Commands UI, this

Liquid Robotics Proprietary and Confidential

Parameter	Description
	parameter allows a command view that hides the expert level commands. A button can be toggled to display all of the commands supported by the Vehicle configuration (the expert commands along with the non-expert commands).
`label`	The name of the command that appears in the UI. For example, in the WGMS SV3 Commands UI, this is the name that appears on the button that executes the command.
`method`	The name of the command that is used in the command syntax.
`paramLabels`	The text associated with a field in the UI where you need to enter information before executing the command. For example, the text associated with the field for the `Force Rudder` command in the Commands Manifest snippet is: `Forced rudder angle in degrees`

`paramTypes`	Defines the numerical value that is acceptable in a field.
	
	`paramTypes` that may appear in the Commands Manifest include:
	
	`'double'`
	`'float'`
	`'integer'`
	`'boolean'`
	`'java.lang.String'`
	`'class [D'`
	
	Where `'java.lang.String'` is a text entry and `'class [D'`, is Longitude and Latitude (`longLat`). `longLat` is expressed by comma separated pairs of decimal numbers. Each `longLat` pair defines a precise location on the surface of the earth. When a series of `longLat` pairs are entered, each pair is separated from the next pair by a comma. For example, a series of 4 `longLat` values would contain eight decimal numbers, each separated from the next by a comma.

`tag`	The name of a group of related commands. For example, in the WGMS SV3 Commands UI, the `navigator` tag associated with the `Force Rudder` command places it under the **navigator** side tab in the UI. If there is no entry for the `tag` parameter, the entry for the `var` parameter can be used for that purpose.
`tooltip`	This is the information that appears in the tool tip that pops up when you hover your cursor over the command button in the UI.
`var`	The name of a group of related commands. Serves the same purpose as the `tag` parameter, but should only be used if there is no entry for the `tag` parameter.
`varNames`	

Parsing Example

In Figure 11.2, the color of the borders surrounding parameters in the configuration and commands manifest snippets, corresponds to the color of the borders surrounding entities in the SV3 Commands Window UI that those parameters populate. For example, devices in the Configuration Manifest determines that the lightbar belongs in the Devices menu in the UI.

From Configuration Manifest .TXP

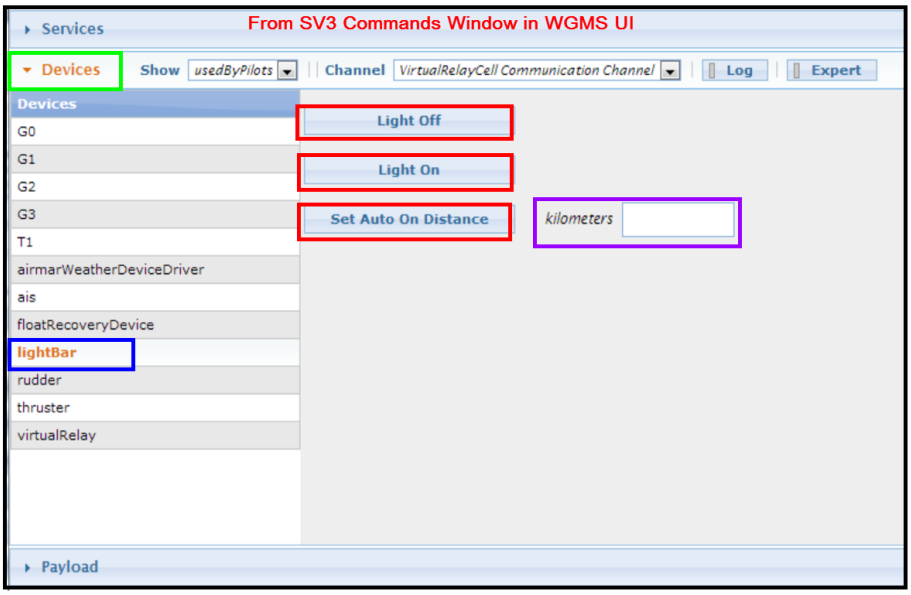
```
"devices": {
    "lightBar": {
        "class": "com.liquidr.regulus.devices.PoweredDevice",
        "enable": true,
        "generateAlarms": false,
        "on": false,
        "reportAmpsErrors": false,
        "reportAmpsWarnings": false,
        "wiring": {
            "connector": "O2Light",
            "domain": "host"
        }
    },
```

From Commands Manifest .TXC

```
{
    "label": "Light On",
    "method": "on",
    "tag": "lightBar",
    "tooltip": "Turn on the mast light",
    "var": "light"
},
{
    "label": "Light Off",
    "method": "off",
    "tag": "lightBar",
    "tooltip": "Turn off the mast light",
    "var": "light"
},
{
    "label": "Set Auto On Distance",
    "method": "setAutoOnDistance",
    "paramLabels": [
        "kilometers"
    ],
    "paramTypes": [
        "double"
    ],
    "tag": "lightBar",
    "tooltip": "If an AIS hit is within this distance,\
               the light will turn on. If it leaves,\
               the light will turn off",
    "var": "light"
},
```

When you hover your cursor over a command button in the WGMS UI, the corresponding tool tip (highlighted in yellow) will appear.

From SV3 Commands Window in WGMS UI

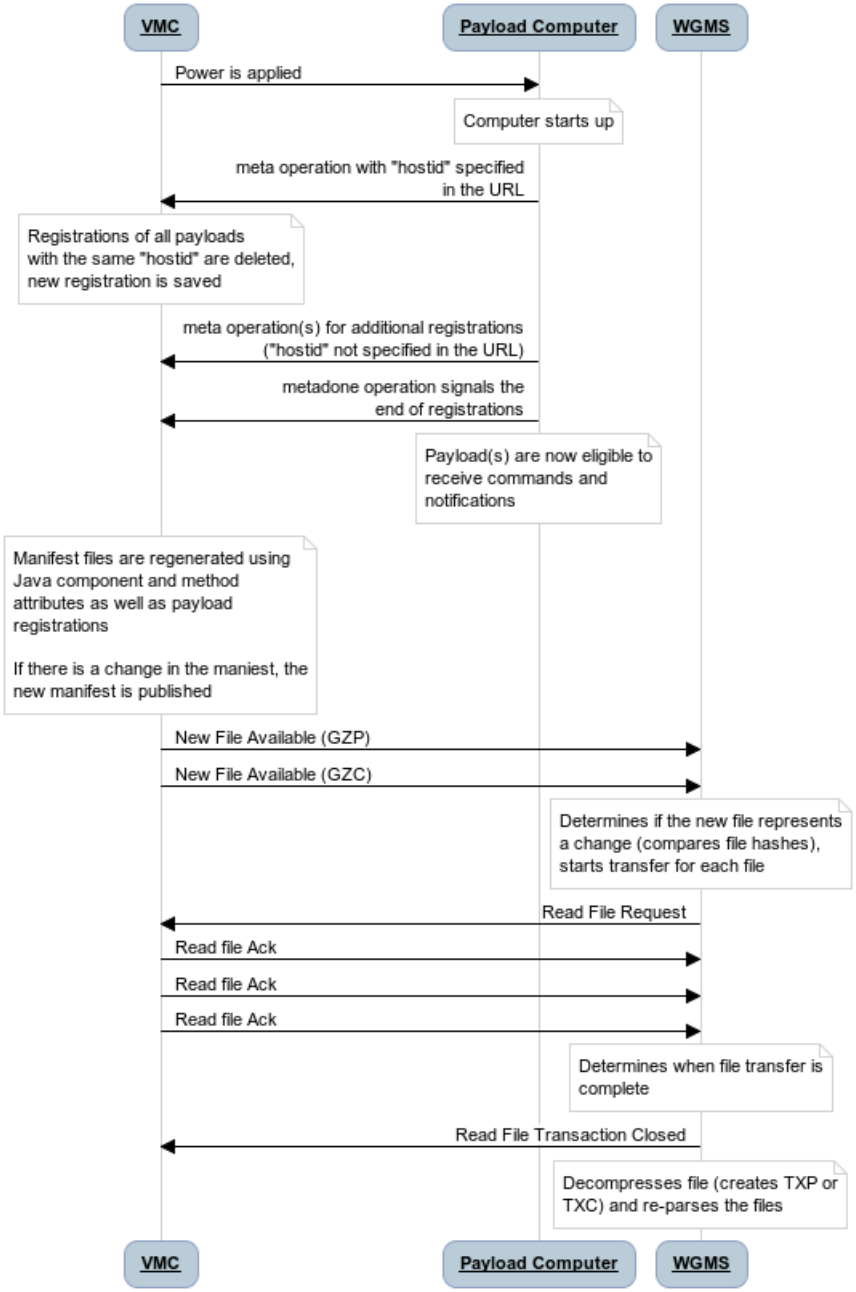


Liquid Robotics Proprietary and Confidential

Expansion Port Payload Registration with Regulus

The SV3 REST payload registration operation (**meta**) includes a JSON fragment that becomes part of the SV3 manifest files. An example of the payload registration operation using WGMS as the remote application is provided in Figure 11.3

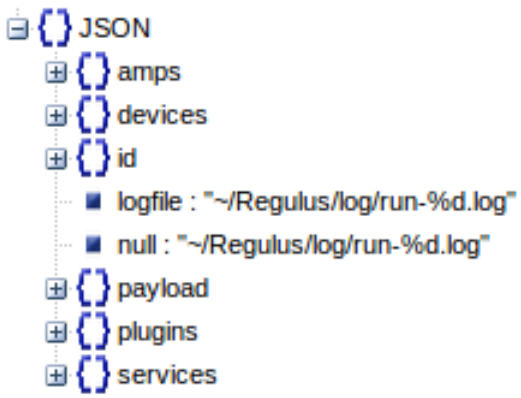
Figure 11.3. SV3 Payload Registration and Manifest Publication



Liquid Robotics Proprietary and Confidential

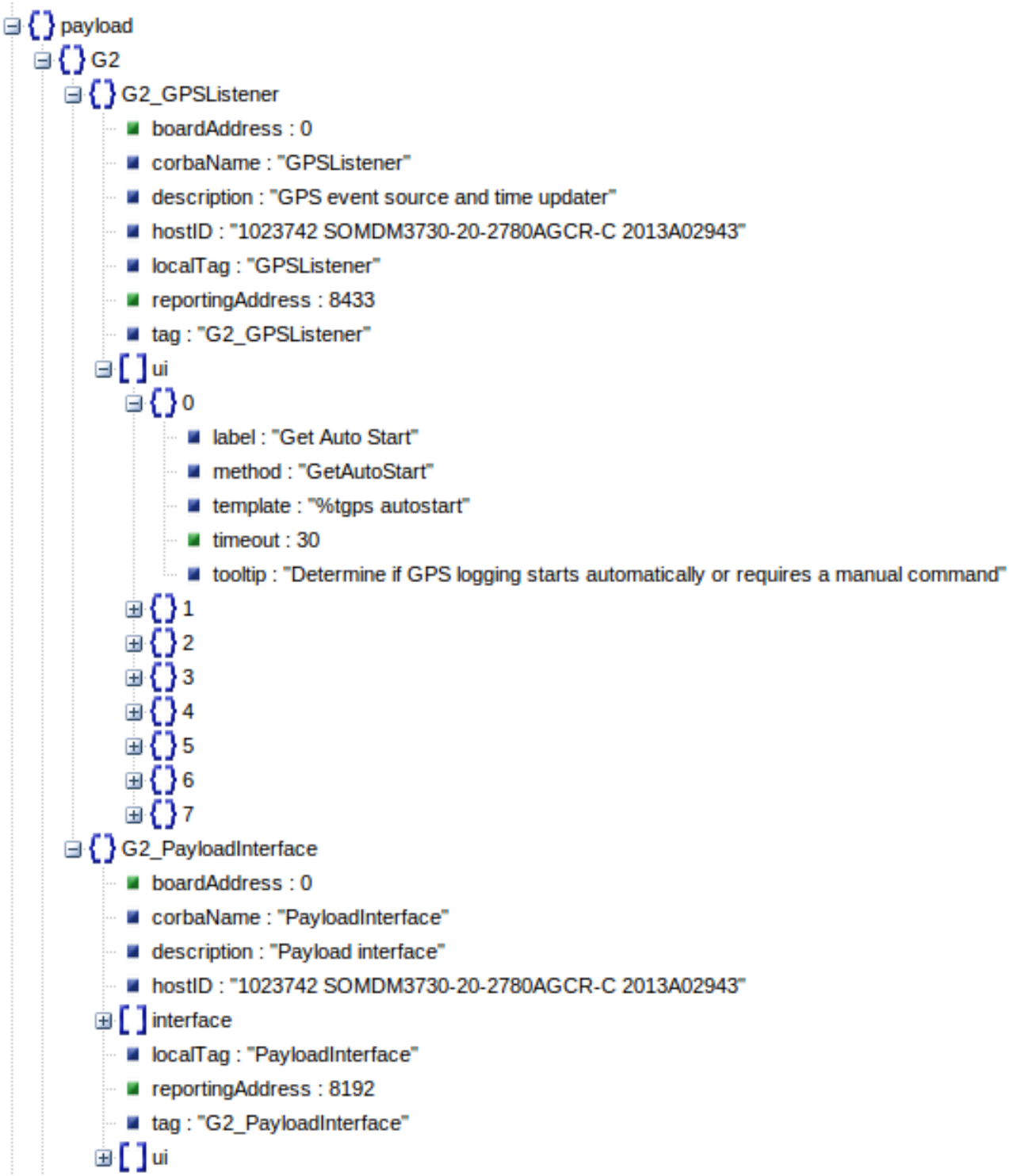
Once the registration is complete (**metadone**), the two manifest files are generated on the Vehicle and named according to the 32-bit unique ID. For example, if the Vehicle's unique ID is 305441741 (which is 0x1234ABCD), the files are named `1234ABCD.GZP` (the Configuration Manifest) and `1234ABCD.GZC` (the Commands Manifest). Once decompressed by the remote application (WGMS in this example), the manifest file names become 1234ABCD.TXP and 1234ABCD.TXC. These files are created as attachments to the Vehicle and appear in the **Attachments** side tab of the WGMS **Vehicle** window.

The TXP file has keys for the various components that the SV3 has access to, and is primarily designed to show how components interconnect. The top-level components are shown here (using http://jsonviewer.stack.hu online JSON viewer):



Under the "payload" key, there are keys for each expansion port that has registered payloads and under each expansion port are keys for each registered payload. Each registered payload key contains a full copy of the meta registration data:

Liquid Robotics Proprietary and Confidential



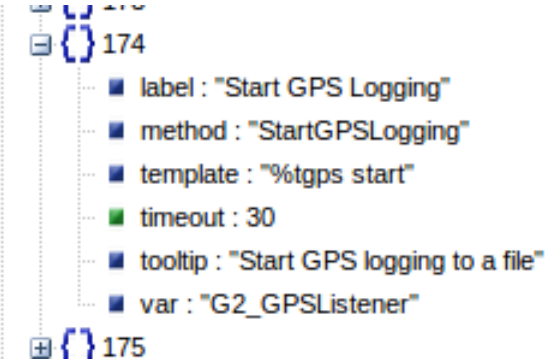
Regulus prefixes the “tag” field with the expansion port name. The original “tag” field is copied to a new field (“localTag”).

Note that even though WGMS knows about certain keys (“boardAddress”, “reportingAddress”, “ui”) and makes use of them, other keys can be added that provide information to users or shore-side automation that WGMS is not aware of. These additional keys are simply ignored by Regulus and WGMS, but are passed through.

Liquid Robotics Proprietary and Confidential

The .TXC file primarily contains commands, though there are keys for vehicle identity and the master list of JSON components:

The “usedByPilots” key is populated with information about each Java class method that can be executed by a pilot, as well as the methods exposed by payloads through the “ui” key in the JSON fragment:



Note that in the case of payload names like “G2_GPSListener”, there is no entry under the .TXC file’s “varNames” key.

To determine if your registration process is working properly:

1. Power on the expansion port your payload is attached to and allow enough time for the payload computer to start up and send its registration commands. Also allow time for the manifest files to be published and transferred. How long this will take depends on the size of the manifests and the speed of the communication channel. For example, two 8K manifest files could take 25 minutes each to transmit over Iridium. The same two 8K manifest files might take a total of 5 minutes over a cellular communications channel. It is important to monitor the file transfer process to confirm that it is continuing (as opposed to stalled-out).

2. Click the WGMS **SV3 Commands** button and verify that the expected payloads and commands appear in the SV3 Commands Window.

If all of the devices, or all of the commands, do not appear in the SV3 Commands Window, review the Regulus web page and the reported manifest files to see if the devices and commands appear there.

If the manifest files appear as expected in the remote application (in this example, as WGMS attachments), but the expected devices and commands are not displayed in the UI (WGMS SV3 Commands Window), then there may be a formatting requirements error (or a bug).

If the manifest files do not appear as expected in the remote application (in this example, as WGMS attachments), the Regulus web page (SV3 Payloads) will indicate whether their JSON registration fragments made it to Regulus:

If the JSON registration fragments made it to Regulus, but not to the remote application (WGMS), then there is a bug.

Liquid Robotics Proprietary and Confidential

If the JSON registration fragments did not make it to Regulus, there is a problem with the payload registration process that you need to investigate and debug.

Regulus Expansion Port REST Interface (XPREST)

Regulus implements a network-based Expansion Port REST interface (XPREST) for payload computers that connect through Expansion Ports. XPREST is implemented using HTTP GET and POST methods and is compatible with the **curl** utility and libraries that are available on many platforms. The client always initiates communications, which is easier to implement and has a lower overhead for payload developers compared to implementing a client side server to receive asynchronous notifications. Future Regulus releases may allow for additional communication scenarios.

XPREST implements most of the features of the serial-based Payload Protocol, including:

- Receiving and responding to command invocations
- Sending telemetry back to shore-side systems
- Sending files back to shore-side systems
- Invoking commands on Regulus
- Receiving notification when the vehicle is about to be shut down

XPREST also provides some new features:

- Registration of individual components on payload computers
- Command descriptions forwarded back to shore-side systems as part of the vehicle manifest
- Forwarding GPS from Regulus
- Parallel access to Regulus from individual components on payload computers (multiple operations can proceed concurrently and don't have to come through a single client side driver)

The following XPREST features are under development:

- Implementation of communication channels
- Forwarding core sensor feeds (AIS, Weather, Water Speed), core sensor telemetry or telemetry from other payloads

Technologies

XPREST utilizes the following technologies:

IP Addresses

Regulus assigns addresses according to the expansion port:

Expansion Port	Connector	IP Address
xp1	G1	192.168.103.X
xp2	G2	192.168.105.X

Liquid Robotics Proprietary and Confidential

xp3	G3	192.168.104.X
xpSub	(Sub)	192.168.101.X

In each case, the DHCP server is configured to begin assigning addresses at 100, and Regulus is configured at address 1. For repeatability, it may be desirable to configure the payload not to use DHCP with a manually assigned address of less than 100.

REST Using CURL

REST (Representational State Transfer) interfaces use HTTP methods like GET to retrieve data from the server, and POST to create resources on the server. REST was chosen primarily because it is easy to implement on the server and the tools to implement its client-side are widely available. There is no need for a web server on the client, so the amount of work to implement a client is not expected to be a barrier to adoption.

With REST interfaces, parameters and small amounts of data can be passed directly in the URL. For example:

http://192.168.103.1/xp?op=poll&id=2000&gps=true

See RFC's 1738 and 3986 for further details on URL and query string formatting and encoding.

The HTTP GET method provides the server with an optional query string, so it can carry some name/value pairs, but it is primarily used to return data from the server to the client. For example, the following **curl** invocation returns data using the sample URL:

```
$ curl --silent 'http://192.168.105.1/xp?op=poll&id=2000&gps=true'
{"gps":{"latitude":37.410943,"longitude":-122.004451,"altitude":"NaN","timeFix":1398964173,
"timeCurrent":1398964173,"speedToGoal":"NaN","sog":"NaN","fog":"NaN"}}
```

The data returned from the server (a JSON fragment containing the next GPS fix received), is returned on standard output.

The HTTP POST method also provides an optional query string, but it can carry attachments. For example, a JSON fragment is sent to the server when a payload device registers. The following example is from the `PayloadInterface` program on the SMC:

```
{"boardAddress":0,
  "corbaName":"PayloadInterface",
  "description":"Payload interface",
  "hostID":"1022102 SOMDM3730-20-2780AGCR-B 0213A00074",
  "interface":[{"swver":"2.1.0"}],
  "reportingAddress":8192,
  "tag":"PayloadInterface",
  "ui":[{"label":"Get Modules",
         "method":"GetModules",
         "paramLabels":["Command"],
         "paramTypes":["java.lang.String"],
         "template":"%tgetmodules",
         "timeout":600,
         "tooltip":"Report the configured modules running on the SMC"},
      {"label":"Run Shell Command",
         "method":"RunShell",
         "paramLabels":["Command"],
```

Liquid Robotics Proprietary and Confidential


```
          "paramTypes":["java.lang.String"],
          "template":"%trunshell %p",
          "timeout":600,
          "tooltip":"Run shell command and return result as a console ack"},
      {"label":"Run Shell (Return File)",
          "method":"RunShellReturnFile",
          "paramLabels":["Command"],
          "paramTypes":["java.lang.String"],
          "template":"file:%trunshell %p",
          "timeout":600,
          "tooltip":"Run shell command and return result as a file transfer"}]}
```

The JSON fragment is passed in a file called *meta.txt*:

```
$ curl -X POST -F 'file=@'"meta.txt" 'http://192.168.103.1/xp?op=meta&id=2000'
OK
```

It is beneficial to bound operations with well-known timeouts. **curl** accepts a command line option (*--max-time*) to limit how long it will wait for a response. In the following example, **curl** will wait for no more than 60 seconds to complete the operation:

```
$ curl --max-time=60 -X POST -F 'file=@meta.txt' 'http://192.168.103.1/xp?op=meta&id=2000'
OK
```

For more information on **curl** see the curl web site at: http://curl.haxx.se, or the Linux man page at: http://linux.about.com/od/commands/l/blcmdl1_curl.htm, or documentation for the version of **curl** that you are using.

REST Using libcurl

libcurl is an open source library that implements many transfer protocols, including HTTP GET and POST. It is available for Linux and Windows from the curl web site at: http://curl.haxx.se The SMC interface library (`SV3RemotingLib`) is based on **libcurl** and **jsoncpp** and is also available as a working sample.

HTTP GET

The following sample code is an example of how you can implement an HTTP GET operation in C++, where the URL is passed as the first argument to the program:

```
#include<iostream>
#include<sstream>
#include<stdexcept>
#include<string>
#include<typeinfo>
#include<vector>
#include<curl/curl.h>

using namespace std;

// ThrowEasy simplifies checking return codes and throws an exception in the case of failure
inline void ThrowEasy(CURLcode code, const std::string& func)
{       if (code != 0) {
                stringstream ss;
                ss << func << " returned " << code << " (" << curl_easy_strerror(code) << ")";
                throw runtime_error(ss.str());
        }
}
```

Liquid Robotics Proprietary and Confidential


```
// SetOpt simplifies setting various options on the CURL instance
// and checking for errors
template<typenameT>
void SetOpt(CURL* curl, CURLoption opt, T t)
{ ThrowEasy(curl_easy_setopt(curl, opt, t), "curl_easy_setopt()"); }

// This function overrides the default behavior and saves the body
// of the response to a string, which can be analyzed after the
// curl invocation completes.
typedef std::vector<std::string> StringArray;
size_t WriteDataFunc(char* ptr, size_t size, size_t nmemb, void* userdata) {
       string& body = *((string*) userdata);
       const size_t newBytes = size*nmemb;
       size_t result;
       try {
              body.insert(body.end(), ptr, ptr+newBytes);
              result = newBytes;
       } catch(...) {
              result = 0;
       }
       return result;
}

// This function overrides the default behavior and saves header
// lines to an array of strings, which can be analyzed later after
// the curl invocation completes
size_t WriteHeaderFunc(void *ptr, size_t size, size_t nmemb, void *userdata) {
       StringArray& headers = *((StringArray*) userdata);
       size_t result;
       try {
              const char* p2 = (const char*)ptr;
              const size_t newBytes = size*nmemb;
              const string header(p2, p2+newBytes);

              headers.push_back(header);
              result = newBytes;
       } catch(...) {
              result = 0;
       }
       return result;
}

int main(int argc, char* argv[])
{
       int result = 0;
       if (argc == 2) {
              CURL* curl = NULL;
              try {
                     StringArray headers;
                     std::string body;
                     if ((curl = curl_easy_init()) == NULL)
                            throw runtime_error("curl_easy_init() failed");
                     const long connectTimeout = 10;
                     const long responseTimeout = 60;
                     const long totalTimeout = 61;
                     const bool verbose = false;
                     SetOpt(curl, CURLOPT_URL, argv[1]);
                     SetOpt(curl, CURLOPT_NOPROGRESS, 1L);
                     SetOpt(curl, CURLOPT_NOSIGNAL, 1L);
                     SetOpt(curl, CURLOPT_CONNECTTIMEOUT, connectTimeout);
```

Liquid Robotics Proprietary and Confidential


```
                        SetOpt(curl, CURLOPT_TIMEOUT, totalTimeout);
                        SetOpt(curl, CURLOPT_FTP_RESPONSE_TIMEOUT, responseTimeout);
                        SetOpt(curl, CURLOPT_VERBOSE, verbose ? 1L : 0L);
                        SetOpt(curl, CURLOPT_WRITEFUNCTION, &WriteDataFunc);
                        SetOpt(curl, CURLOPT_HEADERFUNCTION, &WriteHeaderFunc);
                        SetOpt(curl, CURLOPT_WRITEHEADER, &headers);
                        SetOpt(curl, CURLOPT_WRITEDATA, &body);
                        ThrowEasy(curl_easy_perform(curl), "curl_easy_perform");
                        cout << "Headers=" << endl;
                        for (StringArray::const_iterator ci = headers.begin(); ci !=
headers.end(); ++ci) cout << *ci;
                              cout << "Body=" << endl << body << endl;
                } catch(exception& ex) {
                        cerr << endl
                             << "Caught " << typeid(ex).name() << ": " << ex.what() << endl;
                        result = 1;
                }
                if (curl != NULL) curl_easy_cleanup(curl);
        }
        return result;
}
```

Note that the headers from the invocation are saved in the `headers` string array. The headers should be checked to ensure that an OK status was returned.

HTTP POST

Regulus accepts posts that use the multi-part form method available through **curl**. In addition, posts require and expect `header` and can optionally specify the `MIME` type for the data passed. For example:

```
#include <iostream>
#include <sstream>
#include <stdexcept>
#include <string>
#include <typeinfo>
#include <vector>
#include <curl/curl.h>

using namespace std;
// ThrowEasy simplifies
        checking return codes and throws an exception in the case of failure
inline void ThrowEasy(CURLcode code, const std::string& func)
{
        if (code != 0)          {
                stringstream ss;
                ss << func << " returned " << code << " (" << curl_easy_strerror(code) << ")";
                throw runtime_error(ss.str());
        }
}
// ThrowFormAdd performs the same function for form add return codes
inline void ThrowFormAddError(CURLFORMcode res)
{       if (res != 0)
        {
                stringstream ss;
                ss << "curl_formadd() returned " << res;
                throw runtime_error(ss.str());
        }
}

// SetOpt simplifies setting various options on the CURL instance
```

Liquid Robotics Proprietary and Confidential


```
// and checking for errors
template<typename T>
void SetOpt(CURL* curl, CURLoption opt, T t)
{ ThrowEasy(curl_easy_setopt(curl, opt, t), "curl_easy_setopt()"); }

// This function overrides the default behavior and saves the body
// of the response to a string, which can be analyzed after the
// curl invocation completes.
typedef std::vector<std::string> StringArray;
size_t WriteDataFunc(char* ptr, size_t size, size_t nmemb, void* userdata) {
       string& body = *((string*) userdata);
       const size_t newBytes = size*nmemb;
       size_t result;
       try {
              body.insert(body.end(), ptr, ptr+newBytes);
              result = newBytes;
       } catch(...) {
              result = 0;
       }
       return result;
}

// This function overrides the default behavior and saves header
// lines to an array of strings, which can be analyzed later after
// the curl invocation completes
size_t WriteHeaderFunc(void *ptr, size_t size, size_t nmemb, void *userdata) {
       StringArray& headers = *((StringArray*) userdata);
       size_t result;
       try {
              const char* p2 = (const char*) ptr;
              const size_t newBytes = size*nmemb;
              const string header(p2, p2+newBytes);

              headers.push_back(header);
              result = newBytes;
       } catch(...) {
              result = 0;
       }
       return result;
}

int main(int argc, char* argv[])
{
       int result = 0;
       if (argc == 3) {
              CURL* curl = NULL;
              curl_httppost* formpost = NULL;
              curl_httppost* lastptr = NULL;
              curl_slist* headerlist = NULL;
              try {
                     StringArray headers;
                     std::string body;
                     if ((curl = curl_easy_init()) == NULL) throw
runtime_error("curl_easy_init() failed");
                       headerlist = curl_slist_append(headerlist, "Expect:");
                       const long connectTimeout = 10;
                       const long responseTimeout = 60;
                       const long totalTimeout = 61;
                       const bool verbose = false;
                       SetOpt(curl, CURLOPT_URL, argv[1]);
```

Liquid Robotics Proprietary and Confidential


```
                        SetOpt(curl, CURLOPT_NOPROGRESS, 1L);
                        SetOpt(curl, CURLOPT_NOSIGNAL, 1L);
                        SetOpt(curl, CURLOPT_CONNECTTIMEOUT, connectTimeout);
                        SetOpt(curl, CURLOPT_TIMEOUT, totalTimeout);
                        SetOpt(curl, CURLOPT_FTP_RESPONSE_TIMEOUT, responseTimeout);
                        SetOpt(curl, CURLOPT_VERBOSE, verbose ? 1L : 0L);

                        ////////////////////////////////////
                        /// Adding expect header, optionally ///
                        /// add MIME data type header        ///
                        ////////////////////////////////////
                        SetOpt(curl, CURLOPT_HTTPHEADER, headerlist);
                        ////////////////////////////////////

                        ////////////////////////////////////
                        /// Required for multi-part forms ////
                        ////////////////////////////////////
                        ThrowFormAddError(
                               curl_formadd(&formpost, &lastptr,
                                      CURLFORM_COPYNAME, "file",
                                      CURLFORM_FILE, argv[2],
                                      CURLFORM_END));
                        ThrowFormAddError(
                               curl_formadd(&formpost, &lastptr,
                                      CURLFORM_COPYNAME, "filename",
                                      CURLFORM_COPYCONTENTS, argv[2],
                                      CURLFORM_END));
                        ThrowFormAddError(
                               curl_formadd(&formpost,&lastptr,
                                      CURLFORM_COPYNAME, "submit",
                                      CURLFORM_COPYCONTENTS, "send",
                                      CURLFORM_END));
                        SetOpt(curl, CURLOPT_HTTPPOST, formpost);
                        ////////////////////////////////////

                        ////////////////////////////////////
                        /// Required to save response body ///
                        /// and headers to local variables ///
                        ////////////////////////////////////
                        SetOpt(curl, CURLOPT_WRITEFUNCTION, &WriteDataFunc);
                        SetOpt(curl, CURLOPT_HEADERFUNCTION, &WriteHeaderFunc);
                        SetOpt(curl, CURLOPT_WRITEHEADER, &headers);
                        SetOpt(curl, CURLOPT_WRITEDATA, &body);
                        ////////////////////////////////////

                        ThrowEasy(curl_easy_perform(curl), "curl_easy_perform");

                        cout << "Headers=" << endl;
                        for (StringArray::const_iterator ci = headers.begin(); ci !=
headers.end(); ++ci) cout << *ci;
                        cout << "Body=" << endl << body << endl;
                } catch(exception& ex) {
                        cerr << endl
                             << "Caught " << typeid(ex).name() << ": " << ex.what() << endl;
                        result = 1;
                }
                if (curl != NULL)     curl_easy_cleanup(curl);
                if (formpost != NULL) curl_formfree(formpost);
                if (headerlist != NULL)       curl_slist_free_all(headerlist);
        }
```

Liquid Robotics Proprietary and Confidential


```
    return result;}
```

About JSON

JSON (JavaScript Object Notation; see http://www.json.org/) is a lightweight data-interchange format, easy for both humans and machines to read and write. XPREST passes fragments of JSON to perform registrations and (optionally) to return data from the polling command.

Regulus implements a JSON parser and formatters. The SMC uses the open source `jsoncpp` library, and there are many other implementations that you can use. It is not difficult to implement `jsoncpp` if you can't find another implementation that you prefer.

Functions

The following functions define XPREST operations:

Install JAR (jar)

This HTTP POST message sends a JAR file (Java archive) to Regulus and installs it, allowing the payload to extend the functionality of Regulus. The JAR content is posted.

http://192.168.103.1/xp?op=jar&id=*polling-id*

`id` is a previously-registered polling ID (using the **meta** command)

Payload Registration (meta)

This HTTP POST message registers a polling ID (*`id`*) along with a description of the payload and its commands. The polling ID will be used in all subsequent messages and should be unique for the IP address of the computer. Once registered, Regulus aggregates the description into its manifest files and reports it to the shore-side system. The format of the URL is:

http://192.168.103.1/xp?op=meta&id=*polling-id*&hostid=*registration-group*

The *`hostid`* parameter is optional. Whenever specified, Regulus will look for matching settings in registration data for other payloads and cancel/delete those registrations. The idea is to include the *`hostid`* parameter for the first registration message of a group only. Registrations of other payloads controlled by the same interface will not include it.

Registration data is posted as a JSON fragment:

```
{"boardAddress":0,
  "corbaName":"PayloadInterface",
  "description":"Payload interface",
  "hardware":[{"connector":"ICPwr1"}],
  "hostID":"1022102 SOMDM3730-20-2780AGCR-B 0213A00074",
  "interface":[{"swver":"2.1.0"}],
  "reportingAddress":8192,
  "tag":"PayloadInterface",
  "ui":[{"label":"Get Modules",
        "method":"GetModules",
        "paramLabels":["Command"],
        "paramTypes":["java.lang.String"],
        "template":"%tgetmodules",
        "timeout":600,
```

Liquid Robotics Proprietary and Confidential


```
  "tooltip":"Report the configured modules running on the SMC"},
{"label":"Run Shell Command",
  "method":"RunShell",
  "paramLabels":["Command"],
  "paramTypes":["java.lang.String"],
  "template":"%trunshell %p",
  "timeout":600,
  "tooltip":"Run shell command and return result as a console ack"},
{"label":"Run Shell (Return File)",
  "method":"RunShellReturnFile",
  "paramLabels":["Command"],
  "paramTypes":["java.lang.String"],
  "template":"file:%trunshell %p",
  "timeout":600,
  "tooltip":"Run shell command and return result as a file transfer"}]}
```

The JSON fragment contains the following values that have special meaning (it can also contain fields that are ignored by Regulus, but have meaning for the shore-side system):

- *`boardAddress`* is the 16-bit address that Write Console Script operations are sent to. For expansion ports, this is always set to zero (0).
- *`hardware`* contains an array of elements with the following optional values:
  - *`connector`* names a power switch ("ICPwr1".. "ICPwr8", depending on the installed hardware) associated with the payload box. The `Pwr` function affects the state of the switches.
  - *`manufacturer`* is the name of the manufacturer of the attached device.
  - *`model`* is the model of the attached device.
  - *`serial`* is the serial number of a device attached to the payload. Ideally, this value is determined by communicating with the device. Data reported by the payload may be associated with this serial number.
  - *`swver`* is the software version running on the attached device.
  - *`tty`* is the serial port associated with the payload.
- *`hostID`* should uniquely indicate the registration group that this payload belongs to. There are at least two scenarios for registration groups:
  1. If the payload is implementing its own interface to Regulus and registering on its own, *`hostID`* should uniquely identify the CPU and the payload on that CPU.
  2. If a program is registering multiple payloads, *`hostID`* should uniquely identify the CPU and the registration program.
- *`interface`* is an array of elements associated with the interface software, including the following optional value:
  - *`swver`* is the version string associated with the interface software.
- *`reportingAddress`* is the 16-bit address that is stamped on telemetry coming from this payload.
- *`tag`* is the name that uniquely identifies the payload within the domain of the expansion port.
- *`ui`* is an array of elements that describes the commands that the payload exposes:

Liquid Robotics Proprietary and Confidential

- *label* is a text string that appears on a button in the UI.
- *method* is the name sent back to Regulus when the command is invoked (the method must not include spaces).
- *paramLabels* is an array of strings describing each command parameter.
- *paramTypes* is an array of strings that describe each command parameter type, for example:
  - *boolean* is a logical value (true or false).
  - *int* is a 32-bit signed integer.
  - *float* is a single-precision IEEE 754 floating point value.
  - *double* is a double-precision IEEE 754 float point value.
  - *java.lang.String* is a sequence of characters.
- *template* indicates how to convert the parameters to a command string when sending it back to the payload. The string can contain printf-style format sequences that start with a percent sign (*%*). Two percent signs (*%%*) will expand to a single percent sign. The following format sequences are defined:
  - *%m* The method name
  - *%nnffp* Parameters with optional formatters. If a number (*nn*) is specified, it indicates which parameter to expand; otherwise, all parameters are expanded. Other format characters (*ff*) are as follows:

    * Use spaces to separate parameters (default).

    , Use commas to separate parameters.

    C Quote parameters using C language standards (defaults to shell rules).

    U Quote parameters using URL rules (defaults to shell rules).
  - *%fft* The command tag that must be passed back to reply to the command with optional formatters.
- *tooltip* is a string that provides more information about the command when the mouse hovers over the button (or whenever it would be appropriate).

Payload Registration Complete (metadone)

This HTTP GET message indicates that a payload registration sequence is complete. The resulting manifest will be published to shore-side systems. The format of the URL is:

http://192.168.103.1/xp?op=metadone&id=*polling-id*

Delay Shutdown (offdelay)

This HTTP GET message (typically sent after a shutdown notification from a **poll** message), delays power off to the expansion port by a given number of seconds. The new setting can be optionally set as the default delay.

Liquid Robotics Proprietary and Confidential

http://192.168.103.1/xp?op=offdelay&id=*polling-id*&delay=*nnn*&setdefault=*flag*

`id` is a previously-registered polling ID (using the **meta** command).

`delay` is the length of the delay (number of seconds) before the payload is powered off.

If the `setdefault` parameter is included and set to a non-zero value, the `delay` value will be saved as the default shutdown time.

Poll Regulus (poll)

Using HTTP methods, all communications start with the client. The HTTP GET **poll** message is a way of reading waiting command invocations, GPS data, and shutdown indications. The command includes a timeout value, but the command returns as soon as data is available.

http://192.168.103.1/xp?op=poll&id=*polling-id*&id=*polling-id*$_2$&...&id=*polling-id*$_n$ &format=*nnn*&gps=true&shutdown=true&maxcommands=*ccc*&timeout=*ttt* [0]

`id` is a previously-registered polling ID (using the **meta** command). If multiple polling ID's are given, queued commands for any of the given payloads are returned.

`format` indicates how responses are formatted:

1 (default) = JSON format; commands are assumed to be from the same payload, so no ID (tag) is included, for example:

```
$ curl --silent 'http://192.168.105.1/xp?op=poll&id=2000&format=1&shutdown=true'
{"commands":[{"command":"'YXxutd6L44'runshell 'ls -l'"}]}
```

2 = JSON format; commands can be for multiple payloads and ID's (tags) are included, for example:

```
$ curl --silent 'http://192.168.105.1/xp?op=poll&id=2000&id=20F1&format=2&shutdown=true'
{"commands":[{"id":"PayloadInterface","command":"'YXxutd6L44'runshell 'ls -l'"}]}
```

`gps` indicates whether the next GPS sample is returned (`true`). If `gps` is not specified, or is set to `false`, GPS samples are not returned. The following is an example of a GPS sample:

```
$ curl --silent 'http://192.168.105.1/xp?op=poll&id=2000&id=20F1&format=2&gps=true&shutdown
=true'
{"gps":{"latitude": 37.410782,"longitude":-122.004585,"altitude":"NaN","timeFix":
1399073419,"timeCurrent":1399073419,"seedToGoal":"NaN","sog":"NaN","fog":"NaN"}}
```

Note that `timeFix` is the timestamp from the GPS receiver, while `timeCurrent` was the timestamp on Regulus when the fix was received. Both are in seconds since 1-January 1970 UTC.

`maxcommands` allows the caller to specify how many commands Regulus can return. By default, this value is set to 1. Setting `maxcommands` to 0 (zero) disables returning any commands.

`shutdown` indicates whether client wants to be notified if power to the payload is about to be cut (true). If `shutdown` is not specified, or set to false, no notification is given. The following is an example of being notified that shutdown is about to occur:

```
$ curl --silent 'http://192.168.105.1/xp?op=poll&id=2000&id=20F1&format=2&gps=true&shutdown
=true'
{"gps":{"latitude": 37.410782,"longitude":-122.004585,"altitude":"NaN","timeFix":
1399073419,"timeCurrent":1399073419,"seedToGoal":"NaN","sog":"NaN","fog":"NaN"}}
```

Note that if shutdown notification is received, the client can send a message to delay shutdown long enough for the client computer to gracefully shut down (`offdelay`).

Liquid Robotics Proprietary and Confidential

timeout indicates the number of milliseconds to wait for one of the return conditions to be met. If no conditions are met within the given time, a timeout message is returned:

```
$ curl --silent 'http://192.168.105.1/xp?op=poll&id=2000&id=20F1&format=2&shutdown=true&
timeout=15000'
{"timeout":true}
```

Note that a timeout is not necessarily an error. Typically, a client will specify a relatively short timeout (for example, 15000 milliseconds), and continually issue the command. This places a very light load on Regulus.

If any error conditions occur during execution of the command, an error message will be returned:

```
$ curl --silent 'http://192.168.105.1/xp?op=poll&id=2000&format=2&maxcommands=0&timeout
=1000'
{"error":"Command did not request GPS or a command"}
```

Set AMPS Power Switch (pwr)

This HTTP GET message sets the state of the power switch(es) registered with the **meta** message, or any named power switch.

http://192.168.103.1/xp?op=pwr&id=*polling-id*&cmd=*on-or-off*&sw=*switch-name*

id is a previously-registered polling ID (using the **meta** command).

cmd is the new state of the switch; *on* to turn power on, *off* to turn power off.

sw is an optional parameter indicating which switch(es) to affect (ICPwr1, ICPwr2,...ICPwr8), depending on how many switches exist on the installed hardware. If *sw* is not specified, all of the switch(es) registered with the **meta** message are affected.

Reply to Command (reply)

This HTTP POST message sends reply data for a command previously received using **poll**. The body of the reply is posted:

http://192.168.103.1/xp?op=reply&id=*polling-id*&tag=*ttt*&complete=*flag*&success=*flag*

id is a previously-registered polling ID (using the **meta** command).

tag is the command tag forwarded along with the command (the text returned by the *%t* format sequence in the template for the command).

complete is an optional flag set to true (default) if the command is now complete, or false if there is more data. [This feature is not implemented yet.]

success is an optional flag set to true (default) if the command succeeded, or false if the command failed. [This feature is not implemented yet.]

Run Script (script)

This HTTP GET message allows the payload to run any advertised method on Regulus.

http://192.168.103.1/xp?op=script&id=*polling-id*&cmd=*command-text*

id is a previously-registered polling ID (using the **meta** command).

Liquid Robotics Proprietary and Confidential

cmd is the text of the script invocation, for example:

```
$ curl --silent 'http://192.168.105.1/xp?op=script&id=2000&cmd=uptime.uptime%28%29'
Uptime: 0 days, 07:26:46
```

Send File to Shore as Bulk Data (upload)

This HTTP POST message sends a file to shore using the bulk data service, which is typically restricted to high speed communication channels. The file to send is posted:

http://192.168.103.1/xp?op=upload&id=*polling-id*

id is a previously-registered polling ID (using the **meta** command).

Send File to Shore using WGMS File Transfer Protocol (uploadlegacy)

This HTTP POST message sends a file to shore using the WGMS file transfer protocol over any WGMS communication channel. The file to send is posted:

http://192.168.103.1/xp?op=uploadlegacy&id=*polling-id*

id is a previously-registered polling ID (using the **meta** command).

Send Packet to Shore (packet)

This HTTP POST message sends a level 2 message to shore over the default communications channel, but accepts parameters for some of the fields, and performs the final message assembly. The level 1 message content is posted:

http://192.168.103.1/xp?op=packet&id=*polling-id*&type=*L2-type*&wid=*L2-packet-ID*

id is a previously-registered polling ID (using the **meta** command).

type is the Level 2 type ID (byte 1 of the packet).

wid is the 32-bit Level 2 packet ID (bytes 2-5 of the packet).

Send Raw Packet to Shore (rawpacket)

This HTTP POST message sends a raw level 2 message to shore over the default communications channel. The formatted level 2 message is posted.

http://192.168.103.1/xp?op=rawpacket&id=*polling-id*

id is a previously-registered polling ID (using the **meta** command).

Additional Information on Payload Software Development

For additional information on developing payload-specific Linux applications to run on a System Management Computer (SMC), including design recommendations, a library, setup and installation tools, and a virtual machine for use during development, contact the Liquid Robotics Customer Support organization.

Liquid Robotics Proprietary and Confidential

Customer support is available during regular business hours:

Monday - Friday, 8:00 AM to 5:00 PM (PST).

Telephone:

(US) 888-574-4574

(International) 1-408-636-4205

Email: support@liquidr.com

Chapter 12. Communication Channels for Payload Development and Testing

For a Wave Glider on an extended offshore mission, the most commonly used communications channel is satellite, typically the Iridium Communications Inc. constellation. This works well with the unobstructed view of the sky available from the surface of the sea, but during payload development using satellite communications is impractical since the vehicle is typically indoors. In addition, satellite bandwidth can be expensive and slow compared to other technologies.

To meet this need, the Wave Glider SV3 CCU supports:

- A wired Ethernet connection between the CCU and a LAN
- WiFi
- Cellular Communications

The debug port (D1) on the CCU can be connected to LAN with an Ethernet cable providing direct communications between the CCU and a remote application, for example WGMS. At the same time, the vehicle Web page can be displayed so you can access the information you need to debug. This is the preferred communications method for payload development and testing, when a wired connection is practical.

When a wired connection is not practical, you can use WiFi to display the vehicle Web page while using the Wave Glider SV3 cell modem to support communications between the CCU and the remote application.

Regardless of which communications methods are used for payload development and testing, Liquid Robotics recommends that the Iridium communications channel be exercised prior to launch to be sure it is working properly.

Chapter 13. Troubleshooting

This chapter focuses on problems commonly encountered during **on-shore** Wave Glider payload development. Refer to the *Wave Glider User Manual* for troubleshooting on-water operational problems.

Overview

On-shore troubleshooting falls into two general categories:

1. Fixing non-functioning systems
2. Diagnosing and correcting error conditions

Iridium Interface

In general, most development and systems integration will be done using the WiFi interface. However, it can be helpful to also have the Iridium interface available as an alternate method of communication for troubleshooting.

General Checks

Since the vehicle is fully electric, many problems can be traced back to connectors and/or batteries. The first checks for any problem should be:

1. Check that the Power Key on the CCU is inserted. The Power Key is what turns on the electrical system. An LED light in the Power Key handle is activated when power is on. For more information see “Power Key” (page 34) and “Power Key Indicator Light” (page 35).
2. If the problem appears to be a connectivity issue, check that the electrical connectors are tight. It is possible that a hold down sleeve is cross-threaded or otherwise jammed so a connector is not fully seated. If you're not sure, undo and re-install the connector.
3. Make sure that the electrical cables are plugged into the correct connectors:
   - The Expansion Port connectors are all the same. Cross-connecting should not cause damage, but data communications will be incorrect.
   - The Sensor Port connectors are all the same. Cross-connecting should not cause damage, but data communications will be incorrect.
4. Use WiFi, Cell or Iridium to get the Power Status and verify that the System voltage is at least 14 Volts. If not, the batteries need re-charging. For more information see “External Battery Charging” (page 68).

Re-Booting

Generally, re-booting the CCU is a solution of last resort. Re-booting may temporarily fix the problem, but not address the root cause.

If you have Admin privileges, you can issue the Float Reboot command.

A “hard” re-boot can be initiated by powering the vehicle OFF (remove the Power Key), waiting 20 seconds, and then turning the vehicle back ON (insert the Power Key).

Liquid Robotics Proprietary and Confidential

Contacting Liquid Robotics

If you've tried the suggested solutions and the problem persists, make note of the vehicle identification data and contact Liquid Robotics as described in “Contacting Liquid Robotics, Inc.” (page 264).

Problems and Responses

No Iridium Connection

The Iridium connection is initiated by the vehicle at each Telemetry interval. The interval can be set for an especially long time (as much as 24 hours). Also, if the batteries are low, the vehicle may revert to the "Low Battery" telemetry rate, which is typically longer than the standard rate (to conserve battery power).

Also, Iridium communications require a clear view of the sky to operate. Being near a window is generally not adequate.

Responses:

1. Check that the vehicle power is ON.
2. Check that the Iridium antenna (in the CCU Radome) has a clear view of the sky.
3. Confirm that the vehicle you are attempting to communicate with is the same vehicle you have selected in the remote application.
4. Refresh the remote application Web Page.
5. Use Cell or WiFi communications (if available) to get the IMEI and verify that it is the correct IMEI for the vehicle.
6. Use the remote application to check the strength of the last-sent Telemetry data. The Iridium Signal Strength ranges from 0 to 5. A Level of zero may indicate an Iridium signal problem.

No GPS Reception.

Like Iridium communications, the GPS requires a view of the sky, although it can sometimes get an adequate fix near a large window.

Responses:

1. Check that the vehicle power is ON.
2. Check that the GPS antenna (in the CCU Radome) has a reasonably clear view of the sky.

Power Key Light Does Not Blink

The LED light in the Power Key handle blinks on and off when the system is powered on. If it fails to do this, there may be a CCU electrical or software issue. For more information see “Power Key Indicator Light” (page 35)

No Float Payload Power From CCU

When the system is powered up (or re-booted), power to the payloads is turned OFF. The power needs to be explicitly turned ON. Also, the maximum current that can be supplied to any payload is limited

Liquid Robotics Proprietary and Confidential

(see "Payload Maximum Currents" (page 175)). Any attempt to use more current will result in power to that payload being shut off.

Responses:

1. Check that the payload is connected to the correct connector on the CCU.
2. Check the connectors on CCU and Payload Dry Boxes.
3. Check for an over-current alarm.
4. Check the power status for low batteries.

No Sub Payload or Towfish Payload Power From CCU

As with the Float payloads, Sub and Towfish payload power is Off when the vehicle power is turned On. Also the maximum current that can be supplied to both the Thrudder and the Sub/Towfish payloads combined is 6 Amps.

Responses:

1. Verify that rudder can move. If not, check both the Float and Rudder Module Umbilical connectors. Also check the Umbilical for cracks or tears.
2. Check for an over-current alarm.
3. Check the power status for low batteries.

No CCU-to-Payload RS-232 Communications

Lack of RS-232 communications can be caused by a problem with the CCU electronics, the Payload electronics, and/or the connection between them. If communications were previously established, but no longer work, the best response is to try re-booting the system. The responses below assume that communication has never been established using a specific payload.

Responses:

1. Check for correct cable connection to CCU.
2. Check that CCU has turned on power to the Payload.
3. Check for an over-current alarm.
4. Check payload electronics UART.

Float Continuously Re-Boots

This indicates a serious electronics problem. Contact Liquid Robotics.

Sub Continuously Re-Boots

This indicates a serious electronics problem. Contact Liquid Robotics.

Interpreting CCU Alarms

The Alarm indicator bits are returned in the first two bytes of the Telemetry Type 0x06 data packet. The interpretation for each Alarm bit is:

Liquid Robotics Proprietary and Confidential

0 → No alarm

1 → Alarm set

Table 13.1. CCU Alarms

Alarm	Byte	bit	For More Information
Sub Leak Alarm	0	0	See “Sub Leak Alarm (Byte 0 | bit 0)” (page 261)
Float Leak Alarm	0	1	See “Float Leak Alarm (Byte 0 | bit 1)” (page 261)
Sub-to-Float Comms Alarm	0	2	See “Sub-to-Float Comms Alarm (Byte 0 | bit 2)” (page 262)
Payload Alarm	0	3	See “Payload Alarm (Byte 0 | bit 3)” (page 262)
Float Battery Low Alarm	0	4	See “Float Battery Low Alarm (Byte 0 | bit 4)” (page 262)
Umbilical Fault Alarm	0	5	See “Umbilical Fault Alarm (Byte 0 | bit 5)” (page 262)
GPS Not Functioning	0	7	See “GPS Not Functioning (Byte 0 | bit 7)” (page 262)
Sub Reboot Alarm	1	1	See “Sub Reboot Alarm (Byte 1 | bit 1)” (page 263)
Float-to-Sub Comms Error	1	2	See “Float-to-Sub Comms Error (Byte 1 | bit 2)” (page 263)
Float Temperature Threshold Exceeded	1	4	See “Float Temperature Threshold Exceeded (Byte 1 | bit 4)” (page 263)
Float Pressure Threshold Exceeded	1	5	See “Float Pressure Threshold Exceeded (Byte 1 | bit 5)” (page 263)
EPROM Error	1	6	See “EPROM Error (Byte 1 | bit 6)” (page 263)
Float Reboot Alarm	1	7	See “Float Reboot Alarm (Byte 1 | bit 7)” (page 264)

The alarms are described below:

Sub Leak Alarm (Byte 0 | bit 0)

Water leak was detected in the Thrudder Module. This should never occur unless the vehicle is in the water. The Thrudder Module is factory-sealed and not field-repairable.

Responses:

1. If the Sub is under water, retrieve the vehicle as soon as possible.
2. If the Sub is not under water, try re-booting. If the alarm persists, there is likely an electronics error.

Float Leak Alarm (Byte 0 | bit 1)

Water leak was detected in the CCU. This should never occur unless the vehicle is in the water. The CCU is factory-sealed and not field-repairable.

Responses:

1. If the Float is in the water, retrieve the vehicle as soon as possible.
2. If the Float is out of the water, try re-booting. If the alarm persists, there is likely an electronics problem in the CCU.

Liquid Robotics Proprietary and Confidential

Sub-to-Float Comms Alarm (Byte 0 | bit 2)

The CCU computer detected a data comm error from the Sub.

Responses:

1. Check the electrical connectors on the CCU and the Thrudder Module.
2. Check the Umbilical for cracks or tears.>
3. Try re-booting. If the error occurs again, contact Liquid Robotics.

Payload Alarm (Byte 0 | bit 3)

General problem with a payload. It may be a communications, electrical or environmental issue.

Responses:

1. Check the connectors between the CCU and each MPU dry box.
2. Verify that the payload is not drawing more than the maximum allowed current (see “Payload Maximum Currents” (page 175)).
3. Try re-booting. If the alarm occurs again, contact Liquid Robotics.

Float Battery Low Alarm (Byte 0 | bit 4)

CCU battery level is below the specified "Float Low_Battery Power".

Responses:

1. Power off the vehicle and re-charge the batteries.
2. If the alarm persists after re-charging, contact Liquid Robotics.

Umbilical Fault Alarm (Byte 0 | bit 5)

There is an integrity problem (a leak or broken conductor) on the Umbilical.

Responses:

1. If the vehicle is in the water, retrieve as soon as possible.
2. If the vehicle is out of the water, check the Umbilical for holes or tears.
3. Try re-booting. If the alarm persists, there is likely an electronics error.

GPS Not Functioning (Byte 0 | bit 7)

The vehicle has not received a valid GPS fix for two minutes.

Responses:

1. Verify that the CCU radome has a clear view of the sky.

Liquid Robotics Proprietary and Confidential

2. Try a re-boot, then wait for at least 2 minutes.

Sub Reboot Alarm (Byte 1 | bit 1)

A serious error occurred on the Thrudder Module computer, causing a re-boot.

Responses:

1. Check for a "Sub Leak Alarm" or a "Sub-to-Float Comms Alarm".
2. Try powering OFF, then ON. If the error occurs again, contact Liquid Robotics.

Float-to-Sub Comms Error (Byte 1 | bit 2)

The Sub computer detected a data comm error from the CCU computer.

Responses:

1. Check the electrical connectors on the CCU and the Thrudder Module.
2. Check the Umbilical for cracks or tears.
3. Try re-booting. If the error occurs again, contact Liquid Robotics.

Float Temperature Threshold Exceeded (Byte 1 | bit 4)

The temperature in the CCU is above the "Float Temperature Threshold". If the Float is not exposed to high outdoor temperatures, this indicates an electronics or battery problem.

Responses:

1. Immediately power the vehicle OFF.
2. Check the outside air temperature. If the CCU interior is noticeably warmer than outside, there may be a serious internal problem. Call Liquid Robotics and do not re-power the unit.

Float Pressure Threshold Exceeded (Byte 1 | bit 5)

The pressure in the CCU is above or below the "Float Pressure Threshold" values.

Responses:

1. Remove the valve cover on the top of the CCU, and check the pressure. Adjust as necessary.
2. If the pressure values are within the acceptable range, re-boot and see if the alarm goes off.

EPROM Error (Byte 1 | bit 6)

During boot-up, the system transfers data from the EEPROM to Flash memory, using error checking. This alarm is activated if there is an error in the data transfer.

Responses:

1. Try re-booting. If the error occurs again, contact Liquid Robotics.

Liquid Robotics Proprietary and Confidential

Float Reboot Alarm (Byte 1 | bit 7)

An error occurred on the CCU computer, causing a re-boot.

Responses:

1. If the error occurs continually, contact Liquid Robotics.

Contacting Liquid Robotics, Inc.

If you cannot determine or correct the cause of a problem, contact the Liquid Robotics Support Organization.

To expedite resolving the problem, make a brief list beforehand with the following information:

1. Wave Glider Model Number
2. Any special or added payloads on the vehicle
3. Time/date when the problem occurred
4. Location of vehicle when problem occurred
5. Initial indication of problem
6. Any other indications or unusual readings
7. Attempts at corrective actions and their results

The Liquid Robotics Customer Support organization is available 24/7.

Telephone:

(US) 888-574-4574

(International) 1-408-636-4205

Email: support@liquidr.com

Appendix A. Command Descriptions

Listed below are descriptions and explanations of the CCU commands, and the expected response for each. The ACK/NAK format types are described in Chapter 6: *Vehicle Responses*.

Note: In the following command descriptions all fields marked "reserved" or "not used" must have a value of zero, or the CRC will trigger an error.

Commands for SV3 Vehicles Only

This section provides information on commands that can be issued on Wave Glider SV3 vehicles only. See "SV2 Commands for SV3 Vehicles" (page 268) for commands that can be issued on Wave Glider SV2 and Wave Glider SV3 vehicles.

SV3-only commands are issued using the SV2/SV3 compatible "Write Console Script (0x9100)" (page 277) command by including the SV3 command syntax in the **Data** field of the **Write Console Script** binary command structure.

The **Write Console Script** command delivers a single packet to a 16-bit board address (0,0) on a Wave Glider expecting either ASCII or binary data.

Table A.1. Write Console Script Binary Command Structure

Field Name	Bytes	Value
Structure ID	1	**0x1E** (Write Console Input to Wave Glider)
Command Value	2	**0x9100** (execute component command – ASCII format)
Destination Address	2	0 = Float CCU; otherwise, the address for the board to receive console input or command
Function	1	0 or destination specific value (user entered)
Number of Bytes in Data	2	The number of bytes in the Data field (at least 1)
Data	1..N	Data to send to the Wave Glider CCU console input (ASCII). This is where you enter the SV3-only command syntax. See "SV3-Only Command Syntax" (page 265) for more information.

SV3-Only Command Syntax

This section explains how to use the SV3 Commands Manifest to extract command parameters so you can develop the syntax that must be entered into the **Data** field of the **Write Console Script** binary command structure to issue SV3-Only commands.

See "SV3 Manifests" (page 228) for an explanation of how the SV3 Commands Manifest is organized, definitions of command parameters, and how the manifest is parsed to populate a UI template. You can explore a complete SV3 Commands Manifest in Appendix C: *Commands Manifest Example*.

Liquid Robotics Proprietary and Confidential

The syntax that must be entered into the **Data** field of the **Write Console Script** binary command structure to issue an SV3-Only command always has the same format:

%<var>.<method>(<paramTypes>),

where **var**, **method**, and **paramTypes** are parameters that you can extract from the SV3 Commands Manifest.

The following examples illustrate how you develop the SV3-Only command syntax:

Line Following Command

The SV3 Commands Manifest entry for the **Line Following** command is:

```
{
    "label": "Line Following",
    "method": "setLineFollowing",
    "paramTypes": [
        "boolean"
    ],
    "tag": "navigator",
    "tooltip": "Select 'True' for vehicle to follow straight\
                lines between waypoints. Select 'False' to allow\
                vehicle drift between waypoints",
    "var": "ffn"
},
```

The command syntax that you must enter in the **Write Console Script** command **Data** field is:

```
%ffn.setLineFollowing(boolean)
```

WiFi On Command

The SV3 Commands Manifest entry for the **WiFi On** command is:

```
{
    "label": "WIFI On",
    "method": "turnOn",
    "tag": "wirelessAccessPoint",
    "tooltip": "Turn vehicle wireless access point on",
    "var": "wap"
},
```

The command syntax that you must enter in the **Write Console Script** command **Data** field is:

```
%wap.turnOn()
```

Note that when there is no `paramTypes` information in the manifest entry, you must include a pair of empty parentheses.

Heading Command

The SV3 Commands Manifest entry for the **Heading** command is:

Liquid Robotics Proprietary and Confidential


```
{
    "label": "Heading",
    "method": "heading",
    "paramLabels": [
        "New heading to be used in degrees"
    ],
    "paramTypes": [
        "float"
    ],
    "tag": "navigator",
    "tooltip": "Manually override vehicle heading",
    "var": "ffn"
},
```

The command syntax that you must enter in the **Write Console Script** command **Data** field is:

```
%ffn.heading(float)
```

Hold Custom Command

The SV3 Commands Manifest entry for the **Hold Custom** command is:

```
{
    "label": "Hold Custom",
    "method": "gotoWatch",
    "paramLabels": [
        "Latitude",
        "Longitude",
        "Radius in meters"
    ],
    "paramTypes": [
        "double",
        "double",
        "double"
    ],
    "tag": "navigator",
    "tooltip": "Go to a watch circle",
    "var": "ffn"
},
```

The command syntax that you must enter in the **Write Console Script** command **Data** field is:

```
%ffn.gotoWatch(double,double,double)
```

Note that the multiple `paramTypes` entries are must be comma separated.

Set Low Power Reporting Frequency Command

The SV3 Commands Manifest entry for the **Set Low Power Reporting Frequency** command is:

```
{
    "label": "Set Low Power Reporting Frequency",
    "method": "updateLowBatteryReportRate",
    "paramLabels": [
```

```
            "minutes"
        ],
        "paramTypes": [
            "int"
        ],
        "tag": "telemetry06Reporter",
        "tooltip": "Set the telemetry reporting frequency when power\
                    is low.",
        "var": "telem06"
    },
```

The command syntax that you must enter in the **Write Console Script** command **Data** field is:

```
%telem06.updateLowBatteryReportRate(int)
```

SV2 Commands for SV3 Vehicles

This section lists Wave Glider SV2 commands that can be issued on Wave Glider SV3 Vehicles. See "Commands for SV3 Vehicles Only" (page 265) for commands that can be issued on Wave Glider SV3 vehicles only.

Hold Current Position (0x0001)

- Commands the vehicle to maintain position at a specified Latitude/Longitude.

ACK – Status Return (see "ACK - Status Return (0x11)" (page 95))

NAK – Data Error (see "NAK – Invalid Data (0x22)" (page 101)) if Latitude or Longitude error

Field Name	Bytes	Value
Structure ID	1	**0x10**
Command Value	2	**0x0001**
(reserved)	5	0
Latitude	4	[see "Waypoint Values" (page 85)]
Longitude	4	[see "Waypoint Values" (page 85)]

Hold Station at Waypoint (0x0002)

- Commands the vehicle to move to the specified Target Waypoint and maintain position there.

ACK – Status with Waypoint Read-Back (see "ACK - Status with Waypoint Read-Back (0X10)" (page 94))

NAK – Data Error (see "NAK – Invalid Data (0x22)" (page 101)) if Target Waypoint has not been defined

Field Name	Bytes	Value [range]
Structure ID	1	**0x10**
Command Value	2	**0x0002**
Target Waypoint Number	1	[1 to 254]

Field Name	Bytes	Value [range]
(reserved)	4	0
Latitude	4	0 (not used)
Longitude	4	0 (not used)

Follow Sequential Waypoint Course (0x0003)

- Commands vehicle to move through a contiguous sequence of Waypoints (e.g., 11, 12, 13, 14, 15, 16), specified by using a Begin-Waypoint and an End-Waypoint number (e.g., 11 and 16). The vehicle will repeatedly cycle through the Waypoint sequence until commanded otherwise (e.g., . . . 14, 15, 16, 11, 12, 13, 14, 15, 16, 11, 12, 13, 14, . . .).

- The Target Waypoint is the first Waypoint that the vehicle goes to at the start, and need not be the same as the Begin-Waypoint number. However, the Target Waypoint must be in the sequence.

ACK – No Payload (see "ACK - No Payload Response (0x13)" (page 99))

NAK – Data Error (see "NAK – Invalid Data (0x22)" (page 101)) if missing a Waypoint number in the sequence. For example, if the course is from Waypoint 9 to Waypoint 20, but Waypoint 17 was never defined.

NAK – Data Error (see "NAK – Invalid Data (0x22)" (page 101)) if the Target Waypoint is not within the sequence. For example, if the sequence is from 5 to 18, but the Target Waypoint is 4.

Field Name	Bytes	Value [range]
Structure ID	1	**0x10**
Command Value	2	**0x0003**
Target Waypoint #	1	[within sequence]
Begin Waypoint #	1	[1 to 254]
End Waypoint #	1	[1 to 254]
(reserved)	2	0
Latitude	4	0 (not used)
Longitude	4	0 (not used)

Rudder Right/Left/Center (0x0004, 0x0005, 0x0006)

- Commands the rudder to move either to full left, full right, or to center.

ACK – No Payload (see "ACK - No Payload Response (0x13)" (page 99))

Field Name	Bytes	Value
Structure ID	1	**0x10**
Command Value	2	**0x0004** (Right Rudder) **0x0005** (Left Rudder) **0x0006** (Center Rudder)
(reserved)	5	0
Latitude	4	0 (not used)
Longitude	4	0 (not used)

Liquid Robotics Proprietary and Confidential

Follow Fixed Heading (0x0008)

- Commands the vehicle to immediately proceed on a specified heading from its current position. The heading value is specified in unit degrees.

ACK – No Payload (see "ACK - No Payload Response (0x13)" (page 99))

NAK – Data Error (see "NAK – Invalid Data (0x22)" (page 101)) if the heading value exceeds 359 degrees.

Field Name	Bytes	Value
Structure ID	1	**0x10**
Command Value	2	**0x0008**
(reserved)	3	0
Heading in Degrees	2	[0 to 359]
Latitude	4	0 (not used)
Longitude	4	0 (not used)

Set Rudder Position (0x1006)

- Commands the rudder to move to a position, specified in unit degrees.

ACK – No Payload (see "ACK - No Payload Response (0x13)" (page 99))

Field Name	Bytes	Value [range]	Notes
Structure ID	1	**0x10**	
Command Value	2	**0x1006**	
Position	1	Degrees [–30° to +30°]	[1]
(reserved)	4	0	
Latitude	4	0 (not used)	
Longitude	4	0 (not used)	

Notes:

[1] The sign of the "Position" value is interpreted as follows:

Negative → Left Rudder

Positive → Right Rudder

Visible Light (Marker Light) Off/On (0x2001, 0x2002)

- Turns On or Off the Visible Light at the top of the Weather Station mast (also known as the Marker Light).

ACK – No Payload (see "ACK - No Payload Response (0x13)" (page 99))

Field Name	Bytes	Value
Structure ID	1	**0x10**

Liquid Robotics Proprietary and Confidential

Field Name	Bytes	Value
Command Value	2	**0x2001** (Light Off) **0x2002** (Light On)
(reserved)	5	0
Latitude	4	0 (not used)
Longitude	4	0 (not used)

Retrieve Status (0x4001)

- Retrieves the Status Return data packet. Refer to section 0 for a detailed description of the Status Return packet.

ACK – Status Return (see "ACK - Status Return (0x11)" (page 95))

Field Name	Bytes	Value
Structure ID	1	**0x10**
Command Value	2	**0x4001**
(reserved)	5	0
Latitude	4	0 (not used)
Longitude	4	0 (not used)

Set Status (0x4002)

- Sends new operational parameters to the vehicle. This command has to send many parameters at one time.

ACK – Status Return (see "ACK - Status Return (0x11)" (page 95))

Field Name	Bytes	Value [range] (units)	Notes
Structure ID	1	**0x11**	
Command Value	2	**0x4002**	
(not used)	1	0	
Telemetry Rate	2	[1 – 1440] (minutes)	[1]
Low_Battery Report Rate	2	[1 – 1440] (minutes)	[2]
Alarm Report Rate	2	[1 – 30] (minutes)	[3]
(reserved)	1		
Sub_Float Timeout	1	[20 – 200] (millisecs)	[4]
(reserved)	2	0	
Float Low_Battery Power	1	[1 – 65] (10Wh steps)	[5]
Float Critical_Battery Power	1	[1 – 65] (10Wh steps)	[6]
(reserved)	3	0	
Float Pressure_Threshold High	1	[0 – 255] (kPa)	[7]
Float Pressure_Threshold Low	1	[0 – 255] (kPa)	[7]

Liquid Robotics Proprietary and Confidential

Field Name	Bytes	Value [range] (units)	Notes
Float Temperature_Threshold	1	[0 – 255] (°C)	[8]
Waypoint Arrival Distance	2	[5 – 5000] (meters)	[9]
(reserved)	1	0	
Power Controller Telemetry Rate	2	[5 – 2880] (minutes)	[10]
(reserved)	2	0	
Float-to-Sub Comm Period	1	[1 – 240] (seconds)	[11]
(not used)	8	0	

Notes:

[1] "Telemetry Rate" is the number of minutes between telemetry transmissions.

[2] "Low_Battery Report Rate" telemetry rate is used when battery is below the "Float Low_Battery Power" value (also see note [5]).

[3] "Alarm Report Rate" is the telemetry rate used when an alarm has been triggered.

[4] "Sub_Float Timeout". Failure of the Sub to communicate with the CCU within this time triggers an alarm.

[5] "Float Low_Battery Power". Level at which the battery power is considered "Low."

[6] "Float Critical_Battery Power". Level at which the battery power is considered "Critical."

[7] "Float Pressure_Threshold High/Low". Max and Min allowed CCU Dry Box pressures before an alarm is triggered. Thresholds are adjusted for temperature using the Ideal Gas Law, pV=nRT, referenced to 23°C.

[8] "Float Temperature_Threshold" is the maximum CCU Dry Box temperature allowed before an alarm is triggered.

[9] " Waypoint Arrival Distance" is the distance from the exact Waypoint coordinates at which the vehicle is considered to have "arrived" at the Waypoint.

[10] "Power Controller Telemetry Rate" is the telemetry rate for the Power Controller Status.

[11] "Float-to-Sub Comm Period" is the rate at which the Float sends a command to the Sub.

Retrieve Identification (0x4003)

- Requests the vehicle to send back the ID Packet. Refer to 9.2.3 for a description of the ID Packet format.

ACK – Identification Packet (see "ACK - Vehicle Identification Data (0x12)" (page 96))

Field Name	Bytes	Value
Structure ID	1	**0x10**
Command Value	2	**0x4003**
(reserved)	5	0
Latitude	4	0
Longitude	4	0

Liquid Robotics Proprietary and Confidential

Retrieve Power Status (0x5001)

- Requests the vehicle to send back the Power Controller Status. See "ACK - Power Controller Status (0x14)" (page 99) for a description of the Power Controller Status Packet format.

ACK – Power Controller Packet (see "ACK - Power Controller Status (0x14)" (page 99))

Field Name	Bytes	Value
Structure ID	1	**0x10**
Command Value	2	**0x5001**
(reserved)	5	0
Latitude	4	0
Longitude	4	0

New Waypoint (0x8000)

Waypoint numbers 0 and 255 are not valid for waypoint creation. The ACK should include the number of new waypoints created, the number of previous waypoints overwritten, and as long as at least 1 waypoint has been added or modified by the Wave Glider, it should respond with an ACK. The Wave Glider will NAK if no waypoints in the system have been added or modified.

Field Name	Bytes	Default Value
Structure Id	1	0x10
Command Value	2	0x8000
Field 3	1	0
Field 4	1	0
Field 5	1	0
Field 6	1	0
Waypoint Number	1	Waypoint number
Latitude	4	Latitude
Longitude	4	Longitude

Read Waypoint (0x8003)

- Requests the Lat/Long coordinates of a specific Waypoint number.

ACK – The ACK should be structure type 0x10 which includes the status with the latitude/longitude of the waypoint read in the hold lat/long fields (see "ACK - Status with Waypoint Read-Back (0X10)" (page 94))

NAK – The Wave Glider will NackData if no such Waypoint exists (see "NAK – Invalid Data (0x22)" (page 101)) if the specified Waypoint number hasn't been defined.

Field Name	Bytes	Value [range]
Structure ID	1	0x10
Command Value	2	0x8003

Liquid Robotics Proprietary and Confidential

Field Name	Bytes	Value [range]
Waypoint Number	1	Waypoint number
Field 4	1	0
Field 5	1	0
Field 6	1	0
Field 7	1	0
Latitude	4	0
Longitude	4	0

Note:

When the ACK is returned, remote applications will compare the returned Lat/Long values to the values stored in the database. If they do not agree, an alarm will be issued.

Follow Course and Loop (0x8101)

- Commands the vehicle to follow a course from Waypoint to Waypoint in the order specified. The Waypoint numbers need not be sequential. When the vehicle reaches the last Waypoint, it goes to the first Waypoint and continues looping until terminated by another command.

ACK – No Payload (see “ACK - No Payload Response (0x13)” (page 99))

NAK – Data Error if:

a) Any undefined Waypoint numbers.

b) Count does not match number of Waypoints specified.

c) The Target Waypoint is not one of the course Waypoints specified.

Field Name	Bytes	Value [range]	Notes
Structure ID	1	**0x12**	
Command Value	2	**0x8101**	
Target Waypoint	1	[1 to 254]	[1] [2]
Count of Waypoints	1	[1 to 213]	
Waypoint Number	1	[1 to 254]	
Waypoint Number	1	[1 to 254]	
…..	…	…	
…..	…	…	
Waypoint Number	1	[1 to 254]	

Notes:

[1] The Target Waypoint is the first Waypoint that the vehicle goes to, and must be the same as one of the course Waypoints specified in the command.

[2] If the Target Waypoint is specified more than once in the list of waypoint numbers, the vehicle will initially proceed toward the first course Waypoint in the list.

Liquid Robotics Proprietary and Confidential

Follow Course With Options (0x8102)

- Commands vehicle to follow a course from Waypoint to Waypoint in the order specified. An additional option field specifies whether to loop continuously, or to hold at the last Waypoint.

ACK – No Payload (see "ACK - No Payload Response (0x13)" (page 99))

NAK – Data Error if:

a) Any undefined Waypoint numbers.

b) Count does not match number of Waypoints specified.

c) The Target Waypoint is not one of the course Waypoints specified.

d) The Navigation Option is not 0 or 1.

Field Name	Bytes	Value [range]	Notes
Structure ID	1	**0x12**	
Command Value	2	**0x8102**	
Target Waypoint	1	[1 to 254]	[1]
Navigation Option	1	**0x00** or **0x01**	[2]
Count of Waypoints	1	[1 to 213]	
Waypoint Number	1	[1 to 254]	
Waypoint Number	1	[1 to 254]	
. . . . .	. . .	. . .	
. . . . .	. . .	. . .	
Waypoint Number	1	[1 to 254]	

Notes:

[1] The "Target Waypoint" is the first Waypoint that the vehicle goes to, and must be the same as one of the course Waypoints specified in the command.

[2] The "Navigation Option" indicates what the vehicle should do when it reaches the last Waypoint. The options are:

0x00 → Loop Continuously

0x01 → Hold at Last Waypoint

Read File Request (0x9003)

- Send a request to read a file from the vehicle.

ACK – Read File ACK

NAK:

- Command Error (see "NAK - Invalid Command (0x22)" (page 101))
- Data Error (see "NAK – Invalid Data (0x22)" (page 101))

Liquid Robotics Proprietary and Confidential

- File Error (see "NAK – File Error (0x24)" (page 101))

Field Name	Bytes	Value [range]	Note
Structure ID	1	**0x1B**	
Command Value	2	**0x9003**	
Storage Controller Address	2	[1 to 256] + [1 to 256]	[1]
File Name	11	8+3 ASCII string. (see "File Names" (page 86))	[2]
Transaction ID	4	See note [3]	[3]
Offset (Seek)	4	[0 to (File_Length – 1)]	[4]
Read Length	4	**0** or [1 to File_Length]	[5]
Max Response Size	2	**175**	[6]

Notes:

[1] The "Storage Controller Address" is copied from the New File Available request.

[2] The "File Name" is an ASCII string padded by spaces. (see "File Names" (page 86))

[3] The "Transaction ID" is -1 for the first read. For retries, it is the original Read File ID. (see "Read File Transaction Closed (0x9007)" (page 276))

[4] The "Offset" is the number of bytes from the start of the file at which the write begins. Offset = 0 → Start of File.

[5] The "Read Length" indicates how many bytes to read from the Offset. Read Length = 0 → Read entire file starting from the Offset.

[6] The "Max Response Size" is the maximum number of bytes to allocate for the response. The vehicle may choose to report fewer bytes than requested.

Read File Transaction Closed (0x9007)

- Sends a transaction closed message to the vehicle to signify successful reception of the file. The CCU then removes the file from the vehicle's memory.

ACK – No Payload (see "ACK - No Payload Response (0x13)" (page 99))

NAK – Command Error (see "NAK - Invalid Command (0x22)" (page 101)), Data Error (see "NAK – Invalid Data (0x22)" (page 101))

Field Name	Bytes	Default Value	Notes
Structure ID	1	**0x10**	
Command Value	2	**0x9007**	
Storage Controller Address	2	[1 to 256] + [1 to 256]	[1]
(reserved)	3	0	
File ID	4	remote applications-generated	
Longitude	4	0 (not used)	

Notes:

Liquid Robotics Proprietary and Confidential

[1] The "Storage Controller Address" is the 1-byte Board Address, and the 1-byte Task ID (see “Addresses” (page 84)). The Float CCU has the reserved address value: 0x0000.

Write Console Script (0x9100)

This is a single packet command to a particular 16-bit board address on a Wave Glider expecting either ASCII or binary data.

Field Name	Bytes	Value
Structure ID	1	**0x1E** (Write Console Input to Wave Glider)
Command Value	2	**0x9100** (execute component command – ASCII format)
Destination Address	2	0 = Float CCU; otherwise, the address for the board to receive console input or command
Function	1	0 or destination specific value (user entered)
Number of Bytes in Data	2	The number of bytes in the Data field (at least 1)
Data	1..N	Data to send into Wave Glider Console input (ASCII)

Float Reboot (0xA000)

System Admin/Developer roles have access to this command to reset the float in the case of no-response or debug. This command is not expected to be ACKed by the vehicle and will be followed up by a Float Reboot alarm on the next telemetry transmission.

Field Name	Bytes	Default Value
Structure Id	1	**0x10**
Command Value	2	**0xA000**
Field 3	1	0
Field 4	1	0
Field 5	1	0
Field 6	1	0
Field 7	1	0
Latitude	4	0
Longitude	4	0

Appendix B. Configuration Manifest Example

The following is an example of a Configuration Manifest:


```
{
    "amps": {
        "domains": {
            "host": {
                "manifest": {
                    "Description": "Host Power Domain",
                    "DomainIndex": 1,
                    "DomainName": "Host",
                    "DomainSerial": "0x3332470b3236303200240022",
                    "ParentID": ""
                },
                "wiring": {
                    "Connectors": {
                        "B1": {
                            "module": 16,
                            "port": 1
                        },
                        "B10": {
                            "module": 20,
                            "port": 2
                        },
                        "B2": {
                            "module": 16,
                            "port": 2
                        },
                        "B3": {
                            "module": 17,
                            "port": 1
                        },
                        "B4": {
                            "module": 17,
                            "port": 2
                        },
                        "B5": {
                            "module": 18,
                            "port": 1
                        },
                        "B6": {
                            "module": 18,
                            "port": 2
                        },
                        "B7": {
                            "module": 19,
                            "port": 1
                        },
                        "B8": {
                            "module": 19,
                            "port": 2
                        },
                        "B9": {
                            "module": 20,
```

Liquid Robotics Proprietary and Confidential


```
        "port": 1
    },
    "D": {
        "serial": "Error"
    },
    "G1": {
        "module": 11,
        "port": 256
    },
    "G2": {
        "module": 12,
        "port": 256
    },
    "G3": {
        "module": 8,
        "port": 256
    },
    "ICAIS": {
        "module": 7,
        "port": 4,
        "serial": "LX4"
    },
    "ICCell": {
        "module": 4,
        "port": 6
    },
    "ICEthernetSwitch": {
        "module": 5,
        "port": 3
    },
    "ICIridium": {
        "module": 32,
        "port": 1,
        "serial": "O2"
    },
    "ICUBlox": {
        "module": 32,
        "port": 2,
        "serial": "LX0"
    },
    "ICVMC": {
        "module": 32,
        "port": 3
    },
    "NC1": {
        "module": 4,
        "port": 4
    },
    "NC2": {
        "module": 7,
        "port": 3
    },
    "NC3": {
        "module": 7,
        "port": 5
    },
    "NC4": {
        "module": 7,
        "port": 6
    },
```

Liquid Robotics Proprietary and Confidential


```
"NC5": {
    "module": 5,
    "port": 4
},
"NC6": {
    "module": 5,
    "port": 5
},
"NC7": {
    "module": 5,
    "port": 6
},
"O1SFore": {
    "module": 10,
    "port": 1
},
"O1SMid": {
    "module": 10,
    "port": 2
},
"O1Water": {
    "module": 4,
    "port": 5,
    "serial": "LX5"
},
"O2Light": {
    "module": 4,
    "port": 3
},
"O2Radar": {
    "module": 5,
    "port": 1
},
"O2Recovery": {
    "module": 5,
    "port": 2
},
"O2SAft": {
    "module": 9,
    "port": 1
},
"S1": {
    "module": 7,
    "port": 2,
    "serial": "LX3"
},
"S2": {
    "module": 7,
    "port": 1,
    "serial": "LX2"
},
"S3": {
    "module": 4,
    "port": 2,
    "serial": "LX7"
},
"S4": {
    "module": 4,
    "port": 1,
    "serial": "LX6"
```

Liquid Robotics Proprietary and Confidential


```
                    },
                    "U1": {
                        "module": 15,
                        "port": 257,
                        "serial": "LX1"
                    }
                }
            }
        },
        "thrudder": {
            "manifest": {
                "Description": "Thrudder Power Domain",
                "DomainIndex": 2,
                "DomainName": "",
                "DomainSerial": "0x00000000",
                "ParentID": ""
            },
            "wiring": {
                "Connectors": {
                    "ICRudder": {
                        "module": 1,
                        "port": 2
                    },
                    "ICThruster": {
                        "module": 1,
                        "port": 3
                    },
                    "SUBEXP": {
                        "module": 0,
                        "port": 256
                    }
                }
            }
        }
    }
},
"devices": {
    "airmarWaterspeed": {
        "class": "nmea",
        "enable": true,
        "generateAlarms": false,
        "on": true,
        "reportAmpsErrors": false,
        "reportAmpsWarnings": false,
        "serial": {
            "baudrate": 4800,
            "mode": "rs422"
        },
        "wiring": {
            "connector": "O1Water",
            "domain": "host"
        }
    },
    "airmarWeatherDeviceDriver": {
        "class": "com.liquidr.regulus.devices.AirmarWeatherDeviceDriver",
        "enable": true,
        "generateAlarms": false,
        "on": true,
        "quality": 80,
        "reportAmpsErrors": false,
```

Liquid Robotics Proprietary and Confidential


```
        "reportAmpsWarnings": false,
        "serial": {
            "baudrate": 4800,
            "mode": "rs422"
        },
        "wiring": {
            "connector": "S4",
            "domain": "host"
        }
    },
    "ais": {
        "class": "nmea",
        "enable": true,
        "generateAlarms": false,
        "on": true,
        "reportAmpsErrors": false,
        "reportAmpsWarnings": false,
        "serial": {
            "baudrate": 38400,
            "mode": "rs232"
        },
        "wiring": {
            "connector": "ICAIS",
            "domain": "host"
        }
    },
    "cellModem": {
        "class": "com.liquidr.regulus.devices.PoweredDevice",
        "enable": true,
        "generateAlarms": false,
        "on": true,
        "postPowerOff": "ifdown eth0.108 --force",
        "postPowerOn": "sleep 30;ifup eth0.108",
        "reportAmpsErrors": false,
        "reportAmpsWarnings": false,
        "requires": "ethernetSwitch",
        "startAfter": "deviceManager",
        "wiring": {
            "connector": "ICCell",
            "domain": "host"
        }
    },
    "debugClient": {
        "class": "com.liquidr.regulus.devices.PoweredDevice",
        "enable": false,
        "generateAlarms": false,
        "on": false,
        "postPowerOff": "ifdown --force eth0.102=dhcpclient",
        "postPowerOn": "ifup eth0.102=dhcpclient",
        "reportAmpsErrors": false,
        "reportAmpsWarnings": false,
        "requires": "ethernetSwitch",
        "startAfter": "deviceManager"
    },
    "debugLAN": {
        "class": "com.liquidr.regulus.devices.PoweredDevice",
        "enable": true,
        "generateAlarms": false,
        "on": false,
        "postPowerOff": "ifdown --force eth0.102",
```

Liquid Robotics Proprietary and Confidential


```
        "postPowerOn": "ifup eth0.102",
        "reportAmpsErrors": false,
        "reportAmpsWarnings": false,
        "requires": "ethernetSwitch",
        "startAfter": "deviceManager"
    },
    "debugServer": {
        "class": "com.liquidr.regulus.devices.PoweredDevice",
        "enable": false,
        "generateAlarms": false,
        "on": false,
        "postPowerOff": "ifdown --force eth0.102=dhcpserver",
        "postPowerOn": "ifup eth0.102=dhcpserver",
        "reportAmpsErrors": false,
        "reportAmpsWarnings": false,
        "requires": "ethernetSwitch",
        "startAfter": "deviceManager"
    },
    "ethernetSwitch": {
        "class": "com.liquidr.regulus.devices.PoweredDevice",
        "enable": true,
        "generateAlarms": false,
        "on": true,
        "postPowerOn": "sleep 30",
        "reportAmpsErrors": false,
        "reportAmpsWarnings": false,
        "startAfter": "deviceManager",
        "wiring": {
            "connector": "ICEthernetSwitch",
            "domain": "host"
        }
    },
    "expansionPort0": {
        "boundAmpsDomain": "",
        "class": "com.liquidr.regulus.SV3payloads.SV3PayloadDevice",
        "enable": true,
        "generateAlarms": false,
        "name": "xp0",
        "on": true,
        "payloadPortName": "VMC-Pyld",
        "reportAmpsErrors": false,
        "reportAmpsWarnings": false,
        "smcInitialPower": true
    },
    "expansionPort1": {
        "boundAmpsDomain": "",
        "class": "com.liquidr.regulus.SV3payloads.SV3PayloadDevice",
        "enable": true,
        "generateAlarms": false,
        "ip": "192.168.103.0",
        "name": "xp1",
        "on": false,
        "payloadPortName": "G1",
        "postPowerOff": "ifdown eth0.103",
        "postPowerOn": "sleep 30;ifup eth0.103",
        "reportAmpsErrors": false,
        "reportAmpsWarnings": false,
        "smcInitialPower": true,
        "wiring": {
            "connector": "G1",
```

Liquid Robotics Proprietary and Confidential


```
            "domain": "host"
        }
    },
    "expansionPort2": {
        "boundAmpsDomain": "",
        "class": "com.liquidr.regulus.SV3payloads.SV3PayloadDevice",
        "enable": true,
        "generateAlarms": false,
        "ip": "192.168.105.0",
        "name": "xp2",
        "on": false,
        "payloadPortName": "G2",
        "postPowerOff": "ifdown eth0.105",
        "postPowerOn": "sleep 30;ifup eth0.105",
        "reportAmpsErrors": false,
        "reportAmpsWarnings": false,
        "smcInitialPower": true,
        "wiring": {
            "connector": "G2",
            "domain": "host"
        }
    },
    "expansionPort3": {
        "boundAmpsDomain": "",
        "class": "com.liquidr.regulus.SV3payloads.SV3PayloadDevice",
        "enable": true,
        "generateAlarms": false,
        "ip": "192.168.104.0",
        "name": "xp3",
        "on": false,
        "payloadPortName": "G3",
        "postPowerOff": "ifdown eth0.104",
        "postPowerOn": "sleep 30;ifup eth0.104",
        "reportAmpsErrors": false,
        "reportAmpsWarnings": false,
        "smcInitialPower": true,
        "wiring": {
            "connector": "G3",
            "domain": "host"
        }
    },
    "expansionPortSub": {
        "boundAmpsDomain": "",
        "class": "com.liquidr.regulus.SV3payloads.SV3PayloadDevice",
        "enable": true,
        "generateAlarms": false,
        "ip": "192.168.101.0",
        "name": "xpSub",
        "on": false,
        "payloadPortName": "SUBEXP",
        "postPowerOff": "ifdown eth0.101",
        "postPowerOn": "sleep 30;ifup eth0.101",
        "reportAmpsErrors": false,
        "reportAmpsWarnings": false,
        "smcInitialPower": true,
        "wiring": {
            "connector": "SUBEXP",
            "domain": "thrudder"
        }
    },
```

Liquid Robotics Proprietary and Confidential


```
"floatAhrs": {
    "class": "com.liquidr.regulus.devices.FloatAhrs",
    "enable": true,
    "generateAlarms": false,
    "on": true,
    "reportAmpsErrors": false,
    "reportAmpsWarnings": false,
    "wiring": {
        "connector": {
            "module": 32,
            "port": 4
        },
        "domain": "host"
    }
},
"floatRecoveryDevice": {
    "class": "com.liquidr.regulus.devices.FloatRecoveryDevice",
    "enable": true,
    "generateAlarms": false,
    "on": false,
    "reportAmpsErrors": false,
    "reportAmpsWarnings": false,
    "startAfter": "deviceManager",
    "wiring": {
        "connector": "O2Recovery",
        "domain": "host"
    }
},
"iridiumModem": {
    "class": "com.liquidr.regulus.devices.IridiumModem9603",
    "enable": true,
    "generateAlarms": false,
    "on": true,
    "reportAmpsErrors": false,
    "reportAmpsWarnings": false,
    "serial": {
        "baudrate": 19200,
        "mode": "rs232"
    },
    "startAfter": "deviceManager",
    "wiring": {
        "connector": "ICIridium",
        "domain": "host"
    }
},
"lightBar": {
    "class": "com.liquidr.regulus.devices.PoweredDevice",
    "enable": true,
    "generateAlarms": false,
    "on": false,
    "reportAmpsErrors": false,
    "reportAmpsWarnings": false,
    "wiring": {
        "connector": "O2Light",
        "domain": "host"
    }
},
"payloadPort1": {
    "class": "com.liquidr.regulus.SV2payloads.PayloadPort",
    "enable": true,
```

Liquid Robotics Proprietary and Confidential


```
        "enumerationDelay": "2m",
        "generateAlarms": false,
        "on": false,
        "payloadPortName": "PAYLOAD_1",
        "reportAmpsErrors": false,
        "reportAmpsWarnings": false,
        "serial": {
            "baudrate": 115200,
            "mode": "rs232"
        },
        "startAfter": "deviceManager",
        "wiring": {
            "connector": "S1",
            "domain": "host"
        }
    },
    "payloadPort2": {
        "class": "com.liquidr.regulus.SV2payloads.PayloadPort",
        "enable": true,
        "enumerationDelay": "2m",
        "generateAlarms": false,
        "on": false,
        "payloadPortName": "PAYLOAD_2",
        "reportAmpsErrors": false,
        "reportAmpsWarnings": false,
        "serial": {
            "baudrate": 115200,
            "mode": "rs232"
        },
        "startAfter": "deviceManager",
        "wiring": {
            "connector": "S2",
            "domain": "host"
        }
    },
    "payloadPort3": {
        "class": "com.liquidr.regulus.SV2payloads.PayloadPort",
        "enable": true,
        "enumerationDelay": "2m",
        "generateAlarms": false,
        "on": false,
        "payloadPortName": "PAYLOAD_3",
        "reportAmpsErrors": false,
        "reportAmpsWarnings": false,
        "serial": {
            "baudrate": 115200,
            "mode": "rs232"
        },
        "startAfter": "deviceManager",
        "wiring": {
            "connector": "S3",
            "domain": "host"
        }
    },
    "radarReflector": {
        "class": "com.liquidr.regulus.devices.PoweredDevice",
        "enable": true,
        "generateAlarms": false,
        "on": true,
        "reportAmpsErrors": false,
```

Liquid Robotics Proprietary and Confidential


```
        "reportAmpsWarnings": false,
        "wiring": {
            "connector": "O2Radar",
            "domain": "host"
        }
    },
    "rudder": {
        "alarmReportRateMinutes": 10,
        "class": "com.liquidr.regulus.devices.ReganRudder",
        "enable": true,
        "generateAlarms": false,
        "on": true,
        "reportAmpsErrors": false,
        "reportAmpsWarnings": false,
        "requires": "umbilical",
        "rudderConfigParameters": [
            100,
            3,
            0,
            0,
            25,
            1,
            0,
            0,
            0,
            0,
            0,
            0
        ],
        "rudderGain0": 100,
        "rudderGain50": 50,
        "rudderMax0": 25,
        "rudderMax50": 10,
        "rudderMinMove": 2,
        "rudderPollRate": "1s",
        "rudderTrim": 0,
        "startAfter": "deviceManager",
        "useCompassCalibration": true,
        "wiring": {
            "connector": "ICRudder",
            "domain": "thrudder"
        }
    },
    "smc": {
        "class": "com.liquidr.regulus.devices.PoweredDevice",
        "enable": true,
        "generateAlarms": false,
        "on": true,
        "reportAmpsErrors": false,
        "reportAmpsWarnings": false,
        "startAfter": "deviceManager",
        "wiring": {
            "connector": "ICVMC",
            "domain": "host"
        }
    },
    "thruster": {
        "boostRPM": 300,
        "boostmin": 300,
        "boostpower": 30,
```

Liquid Robotics Proprietary and Confidential


```
        "class": "com.liquidr.regulus.devices.ReganThruster",
        "enable": true,
        "generateAlarms": false,
        "on": true,
        "reportAmpsErrors": false,
        "reportAmpsWarnings": false,
        "requires": "umbilical",
        "startAfter": "deviceManager",
        "thrusterEnabled": true,
        "thrusterPollRate": "15s",
        "wiring": {
            "connector": "ICThruster",
            "domain": "thrudder"
        }
    },
    "torpedoGPS": {
        "class": "com.liquidr.regulus.devices.TorpedoGPS",
        "enable": true,
        "generateAlarms": false,
        "on": true,
        "quality": 65,
        "reportAmpsErrors": false,
        "reportAmpsWarnings": false
    },
    "ubloxDeviceDriver": {
        "class": "com.liquidr.regulus.devices.UbloxDeviceDriver",
        "enable": true,
        "generateAlarms": false,
        "on": true,
        "quality": 100,
        "reportAmpsErrors": false,
        "reportAmpsWarnings": false,
        "serial": {
            "baudrate": 9600
        },
        "wiring": {
            "connector": "ICUBlox",
            "domain": "host"
        }
    },
    "umbilical": {
        "class": "com.liquidr.regulus.devices.PoweredDevice",
        "enable": true,
        "generateAlarms": false,
        "on": true,
        "reportAmpsErrors": false,
        "reportAmpsWarnings": false,
        "startAfter": "deviceManager",
        "wiring": {
            "connector": "U1",
            "domain": "host"
        }
    },
    "virtualRelay": {
        "class": "com.liquidr.regulus.coms.VirtualRelayChannel",
        "enable": true,
        "generateAlarms": false,
        "hostName": "packets.wgms.com",
        "on": true,
        "packetSize": 8192,
```

Liquid Robotics Proprietary and Confidential


```
            "port": 443,
            "reportAmpsErrors": false,
            "reportAmpsWarnings": false,
            "sleepTime": "1m",
            "startAfter": "commsManager",
            "timeout": 30
        }
    },
    "id": {
        "avatar": "http://liquidr.github.io/avatars/DefaultAvatar.png",
        "imei": "300434060722240",
        "lavatar": "",
        "longname": "SV3-030",
        "requires": "liquidr",
        "shortname": "SV3-030",
        "sn": 141824934
    },
    "logfile": "~/Regulus/log/run-%d.log",
    "payload": {
        "xp0": {},
        "xp1": {},
        "xp2": {},
        "xp3": {},
        "xpSub": {}
    },
    "services": {
        "AIS": {
            "enable": true,
            "generateAlarms": false,
            "logHitDistances": false,
            "logPSHI": false
        },
        "AISReporter": {
            "enable": true,
            "generateAlarms": false
        },
        "AMPSManager": {
            "AutoFillSN": "true",
            "generateAlarms": false
        },
        "AMPSPowerWrapper": {
            "generateAlarms": false
        },
        "CPUTempMonitor": {
            "enable": true,
            "generateAlarms": false,
            "period": "1m",
            "periodbias": "0"
        },
        "PDCFilesystem": {
            "enable": true,
            "generateAlarms": false
        },
        "PDCParser": {
            "enable": true,
            "generateAlarms": false
        },
        "SV2PayloadManager": {
            "enable": true,
            "generateAlarms": false
```

Liquid Robotics Proprietary and Confidential


```
},
"SV3PayloadManager": {
    "enable": true,
    "generateAlarms": false
},
"ampsManager": {
    "class": "com.liquidr.regulus.amps.AMPSManager"
},
"bulkDataExport": {
    "bulkhost": [
        "10.10.1.150",
        "10.10.1.200",
        "data.liquidr.com"
    ],
    "enable": true,
    "generateAlarms": false
},
"canTransport": {},
"commandReporter": {
    "enable": true,
    "generateAlarms": false
},
"communicationsManager": {
    "enable": true,
    "generateAlarms": false
},
"configuration": {},
"configurationReporter": {
    "enable": true,
    "generateAlarms": false
},
"deviceManager": {
    "HeadingHost_32_4": {
        "enable": true,
        "generateAlarms": false,
        "on": true,
        "reportAmpsErrors": false,
        "reportAmpsWarnings": false
    }
},
"fileManager": {
    "enable": true,
    "generateAlarms": false
},
"fileSystemCleanup": {
    "enable": true,
    "generateAlarms": false,
    "period": "30m",
    "periodbias": "0"
},
"genericAlarmReporter": {
    "enable": true,
    "generateAlarms": false,
    "reportAlarms": false,
    "reportEvents": true,
    "reportTelemAlarms": true
},
"heading": {
    "generateAlarms": false
},
```

Liquid Robotics Proprietary and Confidential


```
"http": {
    "enable": true,
    "generateAlarms": false,
    "navbar": {
        "9Help/0Readme": "README.mkd",
        "Help/1Contact": "contact.mkd",
        "Help/9Finding Regulus": "Regulus.mkd",
        "Help/9Swag": "http://www.redbubble.com/people/haemishbew/works/10164978\
        -liquid-rowbotics",
        "Help/About SV3 Wavegliders": "http://liquidr.com/technology/sv3.html",
        "Help/Liquid Robotics": "http://liquidr.com",
        "Status/Network": "!netstat -a"
    },
    "port": 80
},
"internetAccessibility": {
    "autostartVPN": true,
    "defaultProvider": "undefined",
    "enable": true,
    "generateAlarms": false,
    "maxAllowablePingFailures": 10,
    "pingInterval": 30,
    "timeBetweenPingAttempts": 30
},
"lightBar": {
    "autoOn": 1,
    "enable": true,
    "generateAlarms": false
},
"linuxState": {
    "generateAlarms": false
},
"linuxWatchdog": {
    "generateAlarms": false
},
"location": {
    "generateAlarms": false,
    "gpsdetaillog": false,
    "logGPGSV": false,
    "timeoutParameter": "20s",
    "watchdogRate": 10
},
"log": {},
"memoryMonitor": {
    "enable": true,
    "generateAlarms": false
},
"navigation": {
    "fallbackLocations": [
        20,
        -155.87169,
        36.86671,
        -121.96609,
        70.64177,
        -160.97168,
        54.01624,
        -166.56235,
        26.76621,
        128.17788,
        -35.1898,
```

Liquid Robotics Proprietary and Confidential


```
174.16386,
-43.74927,
-74.38843,
9.83018,
-84.78149,
-32.1198,
115.50201,
15.42985,
73.7035,
-4.10263,
39.73206,
26.16037,
51.52176,
-33.69292,
18.35747,
26.62229,
34.14997,
-39.43407,
-60.44128,
-8.434,
-34.93893,
28.56523,
-17.93175,
29.08918,
-88.55186,
18.31982,
-65.01743,
41.16392,
-70.19869,
47.54421,
-52.65301,
32.3527,
-64.62227,
37.51987,
-1.13769,
35.96481,
14.54298,
35.50512,
24.28133,
43.02071,
5.9206,
38.64235,
-9.44618,
46.12204,
-1.46496,
50.40195,
-3.43254,
53.78341,
8.42342,
55.10155,
-1.18927,
60.35328,
-0.73162,
60.0213,
24.78859,
64.12829,
-22.15187,
-24.50214,
152.78275,
13.50499,
```

Liquid Robotics Proprietary and Confidential


```
            144.5418
        ],
        "generateAlarms": false,
        "period": "5s",
        "periodbias": "0"
    },
    "navigator": {
        "achieveThreshholdMeters": 25,
        "avoidobstacles": true,
        "boostoff": 1.1,
        "booston": 0,
        "enableLineFollowing": true,
        "generateAlarms": false,
        "lineFollowingStrengthObj": 1,
        "lineFollowingWidth": 25,
        "lookAhead": 5,
        "rudderForceLimit": 5
    },
    "otherSampleService": {
        "command": "pwd",
        "enable": true,
        "generateAlarms": false,
        "restartMinimumInterval": "5s",
        "startAfter": "sampleService"
    },
    "pdc": {
        "class": "com.liquidr.regulus.services.PDCFilesystem",
        "on": true,
        "startAfter": "deviceManager"
    },
    "picture": {
        "enable": true,
        "generateAlarms": false
    },
    "runshell": {
        "enable": true,
        "generateAlarms": false
    },
    "sampleService": {
        "command": [
            "sh",
            "/root/Regulus/conf/sampleService"
        ],
        "enable": true,
        "generateAlarms": false,
        "happyWhen": "Running",
        "restart": "never",
        "startAfter": "deviceManager"
    },
    "scripting": {
        "enable": true,
        "explicitCommands": {},
        "fractoryDefaultsURL": "http://10.10.1.150/FactoryDefaults.script",
        "generateAlarms": false
    },
    "serviceManager": {},
    "setParametersListener": {
        "enable": true,
        "generateAlarms": false
    },
```

Liquid Robotics Proprietary and Confidential


```
"setStatusHandler": {
    "enable": true,
    "generateAlarms": false
},
"speedOverGround": {
    "enable": true,
    "generateAlarms": false
},
"telegraphOffice": {
    "enable": true,
    "generateAlarms": false
},
"telemetry06Reporter": {
    "buildTag": "",
    "enable": true,
    "generateAlarms": false,
    "lowBatteryReportRate": "20m",
    "rate": "5m"
},
"time": {
    "generateAlarms": false
},
"uniqueIDManager": {
    "enable": true,
    "generateAlarms": false
},
"untwist": {
    "enable": true,
    "generateAlarms": false,
    "maxTime": 240,
    "rudderAngle": 45,
    "switchoverAngle": 330,
    "thrustDelay": 7,
    "trustworthy": false
},
"uptime": {
    "enable": true,
    "generateAlarms": false
},
"vehicleHealth": {
    "batteryLevelLow": 200,
    "enable": true,
    "generateAlarms": false,
    "pressureLimitHigh": 130,
    "pressureLimitLow": 105,
    "temperatureLimitHigh": 45,
    "temperatureLimitLow": 5
},
"vehicleID": {
    "generateAlarms": false
},
"waterSpeed": {
    "assumeInTheWater": false,
    "enable": true,
    "generateAlarms": false,
    "inTheWater": false,
    "inconsistencyCountLimit": 10,
    "logWaterSpeedData": false
},
"weatherReporter": {
```

Liquid Robotics Proprietary and Confidential


```
            "enable": true,
            "generateAlarms": false,
            "reportLength": "10m"
        },
        "weatherStation": {
            "enable": true,
            "generateAlarms": false
        },
        "wirelessAccessPoint": {
            "enable": true,
            "generateAlarms": false,
            "wifiPower": true
        }
    }
}
```

Appendix C. Commands Manifest Example

The following is an example of a Commands Manifest:

```
{
    "id": {
        "avatar": "http://liquidr.github.io/avatars/DefaultAvatar.png",
        "imei": "300434060722240",
        "lavatar": "",
        "longname": "SV3-030",
        "requires": "liquidr",
        "shortname": "SV3-030",
        "sn": 141824934
    },
    "usedByPilots": [
        {
            "label": "Vehicle Communications Status",
            "method": "status",
            "returns": "java.lang.String",
            "tag": "communicationsManager",
            "tooltip": "Return the communications manager status",
            "var": "commsManager"
        },
        {
            "expert": true,
            "label": "Clear Default Channel",
            "method": "clearDefaultChannel",
            "tag": "communicationsManager",
            "tooltip": "Clears the default channel set on the vehicle",
            "var": "commsManager"
        },
        {
            "label": "Get Default Channel",
            "method": "getDefaultChannel",
            "returns": "java.lang.String",
            "tag": "communicationsManager",
            "tooltip": "Return the default channel set on the vehicle",
            "var": "commsManager"
        },
        {
            "label": "Set Communications Error Threshold",
            "method": "setCommsErrorThreshold",
            "paramTypes": [
                "int"
            ],
            "tag": "communicationsManager",
            "tooltip": "When a communications channel exceeds this value an alarm will\
                        be reported",
            "var": "commsManager"
        },
        {
            "label": "Turn On",
            "method": "turnOn",
            "tooltip": "Turn this device on",
            "var": "xp3"
```

```
},
{
    "label": "Turn Off",
    "method": "turnOff",
    "tooltip": "Turn this device off",
    "var": "xp3"
},
{
    "label": "Power Expansion Port",
    "method": "powerPortOn",
    "tooltip": "Turn this port on",
    "var": "xp3"
},
{
    "label": "Turn On",
    "method": "turnOn",
    "tooltip": "Turn this device on",
    "var": "xp2"
},
{
    "label": "Turn Off",
    "method": "turnOff",
    "tooltip": "Turn this device off",
    "var": "xp2"
},
{
    "label": "Power Expansion Port",
    "method": "powerPortOn",
    "tooltip": "Turn this port on",
    "var": "xp2"
},
{
    "expert": true,
    "label": "Status",
    "method": "status",
    "returns": "java.lang.String",
    "tooltip": "Get GPS provider information",
    "var": "location"
},
{
    "label": "Turn On",
    "method": "turnOn",
    "tooltip": "Turn this device on",
    "var": "xp1"
},
{
    "label": "Turn Off",
    "method": "turnOff",
    "tooltip": "Turn this device off",
    "var": "xp1"
},
{
    "label": "Power Expansion Port",
    "method": "powerPortOn",
    "tooltip": "Turn this port on",
    "var": "xp1"
},
{
    "label": "Turn On",
    "method": "turnOn",
```

Liquid Robotics Proprietary and Confidential


```
        "tooltip": "Turn this device on",
        "var": "xp0"
    },
    {
        "label": "Turn Off",
        "method": "turnOff",
        "tooltip": "Turn this device off",
        "var": "xp0"
    },
    {
        "label": "Power Expansion Port",
        "method": "powerPortOn",
        "tooltip": "Turn this port on",
        "var": "xp0"
    },
    {
        "expert": true,
        "label": "Clear",
        "method": "clear",
        "tag": "SV3PayloadManager",
        "tooltip": "Clear all pending payload messages",
        "var": "xp"
    },
    {
        "expert": true,
        "label": "Send",
        "method": "send",
        "paramLabels": [
            "Payload Name",
            "Message"
        ],
        "paramTypes": [
            "class java.lang.String",
            "class java.lang.String"
        ],
        "tag": "SV3PayloadManager",
        "tooltip": "Send a script command to a payload",
        "var": "xp"
    },
    {
        "label": "Power Expansion Port",
        "method": "powerPort",
        "paramLabels": [
            "Name of payload, e.g., xp1,xp2,xp3,xpSub"
        ],
        "paramTypes": [
            "class java.lang.String"
        ],
        "tag": "SV3PayloadManager",
        "tooltip": "Power on Expansion Port",
        "var": "xp"
    },
    {
        "label": "Power Integrator Payload",
        "method": "powerPayload",
        "paramLabels": [
            "Name of payload, e.g., xp1,xp2,xp3"
        ],
        "paramTypes": [
            "class java.lang.String"
```

Liquid Robotics Proprietary and Confidential


```
        ],
        "tag": "SV3PayloadManager",
        "tooltip": "Power on Integrator Payload and SMC inside payload",
        "var": "xp"
    },
    {
        "label": "Turn On",
        "method": "turnOn",
        "tooltip": "Turn this device on",
        "var": "xpSub"
    },
    {
        "label": "Turn Off",
        "method": "turnOff",
        "tooltip": "Turn this device off",
        "var": "xpSub"
    },
    {
        "label": "Power Expansion Port",
        "method": "powerPortOn",
        "tooltip": "Turn this port on",
        "var": "xpSub"
    },
    {
        "label": "Deploy",
        "method": "deploy",
        "paramLabels": [
            "recommended value 30, seconds to keep the device powered on"
        ],
        "paramTypes": [
            "int"
        ],
        "returns": "boolean",
        "tag": "floatRecoveryDevice",
        "tooltip": "Float Recovery Device",
        "var": "floatRecovery"
    },
    {
        "expert": true,
        "label": "Resume previous",
        "method": "pop",
        "tag": "navigator",
        "tooltip": "Resume previous course (pop course stack)",
        "var": "ffn"
    },
    {
        "label": "Resume",
        "method": "resume",
        "tag": "navigator",
        "tooltip": "Resume previously suspended navigation settings\
                    - proceed to the next waypoint",
        "var": "ffn"
    },
    {
        "expert": true,
        "label": "Force Rudder",
        "method": "forceRudderProtected",
        "paramLabels": [
            "Forced rudder angle in degrees"
        ],
```

Liquid Robotics Proprietary and Confidential


```
        "paramTypes": [
            "float"
        ],
        "tag": "navigator",
        "tooltip": "Suspend all navigation settings and force the rudder to a fixed angle.\
                    Use 'Resume' to restore navigation settings.",
        "var": "ffn"
    },
    {
        "label": "Line Following",
        "method": "setLineFollowing",
        "paramTypes": [
            "boolean"
        ],
        "tag": "navigator",
        "tooltip": "Select 'True' for vehicle to follow straight lines between waypoints.\
                    Select 'False' to allow vehicle drift between waypoints",
        "var": "ffn"
    },
    {
        "label": "Line Following Status",
        "method": "isLineFollowing",
        "returns": "boolean",
        "tag": "navigator",
        "tooltip": "Returns the state of line following",
        "var": "ffn"
    },
    {
        "label": "Avoid Obstacles",
        "method": "setAvoidObstacles",
        "paramLabels": [
            "Select 'True' to turn on obstacle avoidance"
        ],
        "paramTypes": [
            "boolean"
        ],
        "tag": "navigator",
        "tooltip": "Automatic avoidance of dynamic obstacles - primarily ships.",
        "var": "ffn"
    },
    {
        "label": "Heading",
        "method": "heading",
        "paramLabels": [
            "New heading to be used in degrees"
        ],
        "paramTypes": [
            "float"
        ],
        "tag": "navigator",
        "tooltip": "Manually override vehicle heading",
        "var": "ffn"
    },
    {
        "label": "Resume at closest",
        "method": "resumeAtClosest",
        "tag": "navigator",
        "tooltip": "Resume previously suspended navigation settings\
                    - proceed to the closest waypoint",
        "var": "ffn"
```

Liquid Robotics Proprietary and Confidential


```
},
{
    "expert": true,
    "label": "Resume previous",
    "method": "popToClosest",
    "tag": "navigator",
    "tooltip": "Resume previous course (pop course stack)",
    "var": "ffn"
},
{
    "label": "Arrival Distance",
    "method": "setAchievedThreshholdMeters",
    "paramLabels": [
        "Distance in meters"
    ],
    "paramTypes": [
        "double"
    ],
    "tag": "navigator",
    "tooltip": "Set the distance from a waypoint that the vehicle must achieve to meet\
                the requirement that it has arrived",
    "var": "ffn"
},
{
    "label": "Watch Radius",
    "method": "setRadiusMeters",
    "paramTypes": [
        "double"
    ],
    "tag": "navigator",
    "tooltip": "Sets the radius (meters) of the watch circle",
    "var": "ffn"
},
{
    "expert": true,
    "label": "Watch aspect",
    "method": "setAspect",
    "paramTypes": [
        "double"
    ],
    "tag": "navigator",
    "tooltip": "Watch circle aspect ratio",
    "var": "ffn"
},
{
    "expert": true,
    "label": "Watch rotation",
    "method": "setRotationDegrees",
    "paramTypes": [
        "double"
    ],
    "tag": "navigator",
    "tooltip": "Watch circle rotation in degrees - not useful for circles",
    "var": "ffn"
},
{
    "label": "Figure 8",
    "method": "setFigure8",
    "paramTypes": [
        "boolean"
```

Liquid Robotics Proprietary and Confidential


```
        ],
        "tag": "navigator",
        "tooltip": "Use figure 8 shaped watch circle",
        "var": "ffn"
    },
    {
        "expert": true,
        "label": "Ship Standoff",
        "method": "setStandoffDistanceMeters",
        "paramLabels": [
            "Distance in meters"
        ],
        "paramTypes": [
            "double"
        ],
        "tag": "navigator",
        "tooltip": "Set the distance that the waveglider should leave between it\
                    and any ships.",
        "var": "ffn"
    },
    {
        "expert": true,
        "label": "Set Home",
        "method": "setHome",
        "tag": "navigator",
        "tooltip": "Set the 'home course' to the current course",
        "var": "ffn"
    },
    {
        "expert": true,
        "label": "Go Home",
        "method": "goHome",
        "tag": "navigator",
        "tooltip": "Go to the home course (if set)",
        "var": "ffn"
    },
    {
        "label": "Hold Here...",
        "method": "hold",
        "paramLabels": [
            "Radius of watch circle in meters"
        ],
        "paramTypes": [
            "double"
        ],
        "tag": "navigator",
        "tooltip": "Start a watch circle at the current location",
        "var": "ffn"
    },
    {
        "label": "Hold Here 150M",
        "method": "hold",
        "tag": "navigator",
        "tooltip": "Start a watch circle of radius 150M at the current location",
        "var": "ffn"
    },
    {
        "label": "Hold Custom",
        "method": "gotoWatch",
        "paramLabels": [
```

Liquid Robotics Proprietary and Confidential


```
            "Latitude",
            "Longitude",
            "Radius in meters"
        ],
        "paramTypes": [
            "double",
            "double",
            "double"
        ],
        "tag": "navigator",
        "tooltip": "Go to a watch circle",
        "var": "ffn"
    },
    {
        "label": "Hold Precisely 8",
        "method": "gotoPrecise8",
        "paramLabels": [
            "Latitude",
            "Longitude"
        ],
        "paramTypes": [
            "double",
            "double"
        ],
        "returns": "java.lang.Object",
        "tag": "navigator",
        "tooltip": "Figure 8 style",
        "var": "ffn"
    },
    {
        "label": "Hold Precisely Slalom",
        "method": "gotoPreciseSlalom",
        "paramLabels": [
            "Latitude",
            "Longitude"
        ],
        "paramTypes": [
            "double",
            "double"
        ],
        "returns": "java.lang.Object",
        "tag": "navigator",
        "tooltip": "Slalom skiing style",
        "var": "ffn"
    },
    {
        "label": "Hold Precisely Thruster",
        "method": "gotoPreciseThruster",
        "paramLabels": [
            "Latitude",
            "Longitude"
        ],
        "paramTypes": [
            "double",
            "double"
        ],
        "returns": "java.lang.Object",
        "tag": "navigator",
        "tooltip": "Modulated thruster style",
        "var": "ffn"
```

Liquid Robotics Proprietary and Confidential


```
},
{
    "label": "Hold Precisely Loop",
    "method": "gotoPreciseNone",
    "paramLabels": [
        "Latitude",
        "Longitude"
    ],
    "paramTypes": [
        "double",
        "double"
    ],
    "returns": "java.lang.Object",
    "tag": "navigator",
    "tooltip": "Just head to the point",
    "var": "ffn"
},
{
    "label": "Hold Test Circle",
    "method": "gotoTestCircle",
    "paramLabels": [
        "Latitude",
        "Longitude",
        "Radius in meters"
    ],
    "paramTypes": [
        "double",
        "double",
        "double"
    ],
    "returns": "java.lang.Object",
    "tag": "navigator",
    "tooltip": "Go to a watch circle",
    "var": "ffn"
},
{
    "label": "Hold Randomly",
    "method": "holdRandom",
    "paramLabels": [
        "Latitude",
        "Longitude",
        "Radius in meters"
    ],
    "paramTypes": [
        "double",
        "double",
        "double"
    ],
    "returns": "java.lang.Object",
    "tag": "navigator",
    "tooltip": "Stay within an area, but wander randomly within it\
               (to be unpredictable)",
    "var": "ffn"
},
{
    "expert": true,
    "label": "Set Precise style - int",
    "method": "setPreciseStyle",
    "paramTypes": [
        "int"
```

```
        ],
        "tag": "navigator",
        "var": "ffn"
    },
    {
        "expert": true,
        "label": "Set Precise style - None",
        "method": "setPreciseStyleNone",
        "tag": "navigator",
        "var": "ffn"
    },
    {
        "expert": true,
        "label": "Set Precise style - Slalom",
        "method": "setPreciseStyleSlalom",
        "tag": "navigator",
        "var": "ffn"
    },
    {
        "expert": true,
        "label": "Set Precise style - Thruster",
        "method": "setPreciseStyleThruster",
        "tag": "navigator",
        "var": "ffn"
    },
    {
        "expert": true,
        "label": "BoxK",
        "method": "stdbox",
        "tag": "navigator",
        "tooltip": "Change waypoint list to a standard box at Kawaihae",
        "var": "ffn"
    },
    {
        "expert": true,
        "label": "BoxM",
        "method": "mfgbox",
        "tag": "navigator",
        "tooltip": "Change waypoint list to the standard Sunnyvale manufacturing\
                    checkout box",
        "var": "ffn"
    },
    {
        "expert": true,
        "label": "Clear History",
        "method": "clearHistory",
        "tag": "navigator",
        "tooltip": "Clear list of previous courses",
        "var": "ffn"
    },
    {
        "label": "Auto Power",
        "method": "setBoostThreshholds",
        "paramLabels": [
            "If speed-to-goal drops below this, the thruster is turned on",
            "If speed-to-goal rises above this, the thruster is turned off"
        ],
        "paramTypes": [
            "float",
            "float"
```

Liquid Robotics Proprietary and Confidential


```
        ],
        "tag": "navigator",
        "tooltip": "Change the thresholds at which thruster power is automatically\
                    turned on and off.",
        "var": "ffn"
    },
    {
        "label": "Next Waypoint",
        "method": "advanceToNextWaypoint",
        "tag": "navigator",
        "tooltip": "Skip the current waypoint heading and go on to the next",
        "var": "ffn"
    },
    {
        "label": "Hold Last Waypoint",
        "method": "setHoldLast",
        "paramTypes": [
            "boolean"
        ],
        "tag": "navigator",
        "tooltip": "Select 'True' to hold station at the last waypoint.\
                    Select 'False' to cycle back to the first waypoint.",
        "var": "ffn"
    },
    {
        "expert": true,
        "label": "Line Following Strength",
        "method": "setLineFollowingStrength",
        "paramTypes": [
            "float"
        ],
        "tag": "navigator",
        "tooltip": "Set how strong the vessel tries to stick to the line.\
                    Large values (>2ish) can induce instabilities.",
        "var": "ffn"
    },
    {
        "expert": true,
        "label": "Line Width",
        "method": "setLineWidthMeters",
        "paramTypes": [
            "float"
        ],
        "tag": "navigator",
        "tooltip": "Sets the width of the line to follow.  Narrower makes the line\
                    more precise, but energy can be wasted and instabilities\
                    can be induced.",
        "var": "ffn"
    },
    {
        "label": "Sweep Test",
        "method": "sweepTest",
        "returns": "java.lang.String",
        "tag": "navigator",
        "tooltip": "Sweep the rudder side-to-side to test for jamming",
        "var": "ffn"
    },
    {
        "expert": true,
        "label": "Max turn rate",
```

Liquid Robotics Proprietary and Confidential


```
            "method": "setMaxTurnRateDegreesPerSecond",
            "paramLabels": [
                "Degrees per second"
            ],
            "paramTypes": [
                "float"
            ],
            "tag": "navigator",
            "tooltip": "Set the maximum rate at which the waveglider is allowed to turn",
            "var": "ffn"
        },
        {
            "label": "Light On",
            "method": "on",
            "tag": "lightBar",
            "tooltip": "Turn on the mast light",
            "var": "light"
        },
        {
            "label": "Light Off",
            "method": "off",
            "tag": "lightBar",
            "tooltip": "Turn off the mast light",
            "var": "light"
        },
        {
            "label": "Set Auto On Distance",
            "method": "setAutoOnDistance",
            "paramLabels": [
                "kilometers"
            ],
            "paramTypes": [
                "double"
            ],
            "tag": "lightBar",
            "tooltip": "If an AIS hit is within this distance, the light will turn on.\
If it leaves, the light will turn off",
            "var": "light"
        },
        {
            "label": "Set Reporting Frequency",
            "method": "updateReportRate",
            "paramLabels": [
                "minutes"
            ],
            "paramTypes": [
                "int"
            ],
            "tag": "telemetry06Reporter",
            "tooltip": "Set the telemetry reporting frequency.",
            "var": "telem06"
        },
        {
            "label": "Set Low Power Reporting Frequency",
            "method": "updateLowBatteryReportRate",
            "paramLabels": [
                "minutes"
            ],
            "paramTypes": [
                "int"
```

Liquid Robotics Proprietary and Confidential


```
        ],
        "tag": "telemetry06Reporter",
        "tooltip": "Set the telemetry reporting frequency when power is low.",
        "var": "telem06"
    },
    {
        "label": "Send Telem06 Message",
        "method": "sendTelemetry",
        "tag": "telemetry06Reporter",
        "tooltip": "Force the vehicle to send a Telemetry 06 message to shore.",
        "var": "telem06"
    },
    {
        "expert": true,
        "label": "Set Generic Alarm Reporting",
        "method": "setAlarms",
        "paramTypes": [
            "boolean"
        ],
        "tag": "genericAlarmReporter",
        "tooltip": "Toggle the state of reporting alarms",
        "var": "gar"
    },
    {
        "expert": true,
        "label": "Set Generic Event Reporting",
        "method": "setEvents",
        "paramTypes": [
            "boolean"
        ],
        "tag": "genericAlarmReporter",
        "tooltip": "Toggle the state of reporting events",
        "var": "gar"
    },
    {
        "expert": true,
        "label": "Set Telem06 Alarm Reporting",
        "method": "setTelemAlarms",
        "paramTypes": [
            "boolean"
        ],
        "tag": "genericAlarmReporter",
        "tooltip": "Toggle the state of reporting telem06 alarms",
        "var": "gar"
    },
    {
        "label": "Status",
        "method": "status",
        "returns": "java.lang.String",
        "tooltip": "Provides the heading and measured position",
        "var": "rudder"
    },
    {
        "expert": true,
        "label": "Cell Ping Interval",
        "method": "setPingInterval",
        "paramLabels": [
            "Interval in minutes"
        ],
        "paramTypes": [
```

Liquid Robotics Proprietary and Confidential


```
            "int"
        ],
        "tag": "internetAccessibility",
        "tooltip": "Set the interval between cell modem pings to determine\
                    if the vehicle is in cellular communications range.",
        "var": "internet"
    },
    {
        "expert": true,
        "label": "Ping",
        "method": "operatorPingInternet",
        "tag": "internetAccessibility",
        "tooltip": "Attempt to reach the internet",
        "var": "internet"
    },
    {
        "expert": true,
        "label": "Set Internet Provider",
        "method": "setProvider",
        "paramLabels": [
            "Options are 'cell', 'debug', or 'disabled'."
        ],
        "paramTypes": [
            "class java.lang.String"
        ],
        "returns": "java.lang.String",
        "tag": "internetAccessibility",
        "tooltip": "Sets the default internet connection for the VMC",
        "var": "internet"
    },
    {
        "expert": true,
        "label": "Set Internet Provider",
        "method": "setProvider",
        "paramLabels": [
            "Options are 'cell', 'debug', or 'disabled'.",
            "VPN Toggle Switch"
        ],
        "paramTypes": [
            "class java.lang.String",
            "boolean"
        ],
        "returns": "java.lang.String",
        "tag": "internetAccessibility",
        "tooltip": "Sets the default internet connection for the VMC,\
                    current options are 'cell' and 'debug'",
        "var": "internet"
    },
    {
        "expert": true,
        "label": "Enable / Disable VPN",
        "method": "setVpn",
        "paramTypes": [
            "boolean"
        ],
        "tag": "internetAccessibility",
        "tooltip": "VPN toggle switch",
        "var": "internet"
    },
    {
```

Liquid Robotics Proprietary and Confidential


```
        "expert": true,
        "label": "Get Linux Interface IP Address",
        "method": "getInterfaceIP",
        "paramLabels": [
            "Name of Linux Interface"
        ],
        "paramTypes": [
            "class java.lang.String"
        ],
        "returns": "java.lang.String",
        "tag": "internetAccessibility",
        "var": "internet"
    },
    {
        "expert": true,
        "label": "Get Vehicle VPN IP Address",
        "method": "vpnAddress",
        "returns": "java.lang.String",
        "tag": "internetAccessibility",
        "var": "internet"
    },
    {
        "label": "WIFI On",
        "method": "turnOn",
        "tag": "wirelessAccessPoint",
        "tooltip": "Turn vehicle wireless access point on",
        "var": "wap"
    },
    {
        "label": "WIFI Off",
        "method": "turnOff",
        "tag": "wirelessAccessPoint",
        "tooltip": "Turn vehicle wireless access point off",
        "var": "wap"
    },
    {
        "expert": true,
        "label": "Update VR Packet Size",
        "method": "updatePacketSize",
        "paramLabels": [
            "bytes"
        ],
        "paramTypes": [
            "int"
        ],
        "var": "virtualRelay"
    },
    {
        "expert": true,
        "label": "Update VR Call In Time",
        "method": "updateSleepTime",
        "paramLabels": [
            "seconds"
        ],
        "paramTypes": [
            "class java.lang.String"
        ],
        "var": "virtualRelay"
    },
    {
```

Liquid Robotics Proprietary and Confidential


```
        "label": "Morse Code",
        "method": "tapMessage",
        "paramLabels": [
            "The message to send"
        ],
        "paramTypes": [
            "class java.lang.String"
        ],
        "tag": "telegraphOffice",
        "tooltip": "Flash morse code message through the vehicle marker light",
        "var": "morse"
    },
    {
        "label": "Set Weather Report Length",
        "method": "setReportLength",
        "paramLabels": [
            "minutes"
        ],
        "paramTypes": [
            "class java.lang.String"
        ],
        "tooltip": "Sets the report interval for the weather station ",
        "var": "weatherReporter"
    },
    {
        "label": "Set Special",
        "method": "setSpecialMMSI",
        "paramLabels": [
            "MMSI code of the special ship (the Po'okela is 338149019)"
        ],
        "paramTypes": [
            "int"
        ],
        "tooltip": "Set the MMSI code of the ship to be treated specially;\
                    defaults to the Po'okela",
        "var": "ship"
    },
    {
        "label": "Normal",
        "method": "setTreatmentNormal",
        "tooltip": "Treat the ship selected for special treatment just like\
                    any other (ie. no special treatment)",
        "var": "ship"
    },
    {
        "label": "Ignore",
        "method": "setTreatmentIgnore",
        "tooltip": "Ignore the ship selected for special treatment",
        "var": "ship"
    },
    {
        "label": "Circle",
        "method": "setTreatmentCircle",
        "tooltip": "Circle the ship with a radius of 20m",
        "var": "ship"
    },
    {
        "expert": true,
        "label": "Generate Command File",
        "method": "reportCommands",
```

Liquid Robotics Proprietary and Confidential


```
        "returns": "boolean",
        "tooltip": "Generates the command file with SV3 commands,\
                    will send to shore if necessary",
        "var": "commandReporter"
    },
    {
        "expert": true,
        "label": "Generate Config File",
        "method": "makeConfigFile",
        "returns": "boolean",
        "tag": "configurationReporter",
        "tooltip": "Generates the config file with SV3 configuration,\
                    will send to shore if necessary",
        "var": "configReporter"
    },
    {
        "expert": true,
        "label": "Float reboot",
        "method": "handleFloatReboot",
        "tag": "AMPSManager",
        "tooltip": "Do a full float reboot, including Regulus and Linux",
        "var": "amps"
    },
    {
        "expert": true,
        "label": "Reset rudder port",
        "method": "resetRudder",
        "tag": "AMPSManager",
        "tooltip": "Reset the rudder port",
        "var": "amps"
    },
    {
        "label": "Hardware Status",
        "method": "status",
        "returns": "java.lang.String",
        "tag": "AMPSManager",
        "tooltip": "Obtains relevant hardware manager status",
        "var": "amps"
    },
    {
        "expert": true,
        "label": "Clear Twist Indication",
        "method": "twistIsClear",
        "tooltip": "Doesn't do any untwisting, it just clears the indicator",
        "var": "untwist"
    },
    {
        "expert": true,
        "label": "Untwist CW",
        "method": "untwistCW",
        "tooltip": "Untwist by rotating ClockWise",
        "var": "untwist"
    },
    {
        "expert": true,
        "label": "Untwist CCW",
        "method": "untwistCCW",
        "tooltip": "Untwist by rotating Counter-ClockWise",
        "var": "untwist"
    },
```

Liquid Robotics Proprietary and Confidential


```
{
    "expert": true,
    "label": "Get Twist Indication",
    "method": "status",
    "returns": "java.lang.String",
    "tooltip": "Get the (experimental) indication of umbilical twists",
    "var": "untwist"
},
{
    "expert": true,
    "label": "Set Pressure High Limit",
    "method": "setPressureLimitHigh",
    "paramLabels": [
        "kPa"
    ],
    "paramTypes": [
        "float"
    ],
    "var": "vehicleHealth"
},
{
    "expert": true,
    "label": "Set Pressure Low Limit",
    "method": "setPressureLimitLow",
    "paramLabels": [
        "kPa"
    ],
    "paramTypes": [
        "float"
    ],
    "var": "vehicleHealth"
},
{
    "expert": true,
    "label": "Set Temperature High Limit",
    "method": "setTemperatureLimitHigh",
    "paramLabels": [
        "degrees C"
    ],
    "paramTypes": [
        "float"
    ],
    "var": "vehicleHealth"
},
{
    "expert": true,
    "label": "Set Temperature Low Limit",
    "method": "setTemperatureLimitLow",
    "paramLabels": [
        "degrees C"
    ],
    "paramTypes": [
        "float"
    ],
    "var": "vehicleHealth"
},
{
    "expert": true,
    "label": "Set Battery Low Limit",
    "method": "setBatteryLevelLow",
```

```
        "paramLabels": [
            "watt hours"
        ],
        "paramTypes": [
            "double"
        ],
        "var": "vehicleHealth"
    },
    {
        "expert": true,
        "label": "Reset to factory defaults",
        "method": "resetToFactoryDefaults",
        "returns": "java.lang.String",
        "tooltip": "Resets a variety of tuning parameters to\
                    the factory-suggested values.  Use wisely!",
        "var": "scripting"
    },
    {
        "label": "Vehicle System Uptime",
        "method": "uptime",
        "returns": "java.lang.String",
        "var": "uptime"
    },
    {
        "label": "Send AMPS Power On Command",
        "method": "on",
        "paramLabels": [
            "Domain Name",
            "Module/Slot #",
            "Port #"
        ],
        "paramTypes": [
            "class java.lang.String",
            "int",
            "int"
        ],
        "returns": "java.lang.String",
        "tag": "AMPSPowerWrapper",
        "tooltip": "Power ON AMPS Domain::Module::Port",
        "var": "pwrCtl"
    },
    {
        "label": "Send AMPS Power Off Command",
        "method": "off",
        "paramLabels": [
            "Domain Name",
            "Module/Slot #",
            "Port #"
        ],
        "paramTypes": [
            "class java.lang.String",
            "int",
            "int"
        ],
        "returns": "java.lang.String",
        "tag": "AMPSPowerWrapper",
        "tooltip": "Power OFF AMPS Domain::Module::Port",
        "var": "pwrCtl"
    },
    {
```

Liquid Robotics Proprietary and Confidential


```
        "label": "Assume In Water",
        "method": "setAssumeInWater",
        "paramTypes": [
            "boolean"
        ],
        "tooltip": "If set to 'True' the CCU assumes that the vehicle is in the water,\
                    regardless of any conflicting sensor input.",
        "var": "waterSpeed"
    },
    {
        "label": "Spin Thruster",
        "method": "spinThruster",
        "paramLabels": [
            "power (watts)"
        ],
        "paramTypes": [
            "float"
        ],
        "tooltip": "Set Thruster Power",
        "var": "thruster"
    },
    {
        "expert": true,
        "label": "Thruster Battery Limit",
        "method": "setBoostBatteryCapacityMinimum",
        "paramLabels": [
            "battery level (watt-hours)"
        ],
        "paramTypes": [
            "int"
        ],
        "tooltip": "Set minimum battery capacity required to engage thruster\
                    - in watt-hours, thruster will be turned off at and\
                    below this level",
        "var": "thruster"
    },
    {
        "label": "Enable Thruster for Operation",
        "method": "enableThruster",
        "tooltip": "Allows the operator and navigation to use the thruster",
        "var": "thruster"
    },
    {
        "label": "Disable Thruster",
        "method": "disableThruster",
        "tooltip": "The thruster will never be turned on",
        "var": "thruster"
    },
    {
        "expert": true,
        "label": "Set Thruster Motor Advance",
        "method": "setAdvance",
        "paramLabels": [
            "Advance value",
            "EEPROM Flag"
        ],
        "paramTypes": [
            "int",
            "boolean"
        ],
```

Liquid Robotics Proprietary and Confidential


```
        "returns": "boolean",
        "tooltip": "Set the advance value and write to EEPROM flag",
        "var": "thruster"
    },
    {
        "label": "Status",
        "method": "status",
        "returns": "java.lang.String",
        "tooltip": "Get the instantaneous RPM and Power of the thruster",
        "var": "thruster"
    },
    {
        "label": "Cruise",
        "method": "cruisingSpeed",
        "tooltip": "35 watts 450 rpms",
        "var": "thruster"
    },
    {
        "expert": true,
        "label": "Command Thruster",
        "method": "commandThruster",
        "paramLabels": [
            "power (watts)",
            "rpms"
        ],
        "paramTypes": [
            "float",
            "float"
        ],
        "tooltip": "Set both the power and rpm of the thruster",
        "var": "thruster"
    },
    {
        "label": "Fast",
        "method": "fastestSpeed",
        "tooltip": "50 watts 450 rpms",
        "var": "thruster"
    },
    {
        "label": "Stop",
        "method": "stopThruster",
        "tooltip": "Stop the thruster and turn off port",
        "var": "thruster"
    },
    {
        "label": "Slow",
        "method": "slow",
        "tooltip": "20 watts 450 rpms",
        "var": "thruster"
    },
    {
        "label": "Turn Device On / Off",
        "method": "setOn",
        "paramTypes": [
            "boolean"
        ],
        "tag": "airmarWeatherDeviceDriver",
        "var": "airmarWeather"
    },
    {
```

Liquid Robotics Proprietary and Confidential


```
            "expert": true,
            "label": "Send Enable Heading Message",
            "method": "enableHdtMsg",
            "tag": "airmarWeatherDeviceDriver",
            "var": "airmarWeather"
        }
    ],
    "varNames": {
        "AISReporter": "com.liquidr.regulus.wgmsinterop.AISReporter",
        "CPUTempMonitor": "com.liquidr.regulus.services.CPUTempMonitor",
        "Distance2Force": "com.liquidr.regulus.nav.Distance2Force$Delegate",
        "Eval": "com.liquidr.regulus.eval.Eval",
        "Forcefield": "com.liquidr.regulus.nav.Forcefield$Watch8Attractor",
        "GPSTorpedoGPS": "com.liquidr.regulus.devices.NMEAHandlers.NMEALocationDevice",
        "GPSUblox": "com.liquidr.regulus.devices.NMEAHandlers.NMEALocationDevice",
        "GroundTrack": "com.liquidr.regulus.nav.GroundTrack",
        "HeadingHost_32_4": "com.liquidr.regulus.devices.HeadingDevice",
        "InteractableObservableSet": "com.liquidr.regulus.web.InteractableObservableSet",
        "Log": "com.liquidr.regulus.logging.Log$1",
        "MicroStateReporters": "com.liquidr.regulus.web.MicroStateReporters",
        "OpenAMAccess": "com.liquidr.openam.OpenAMAccess",
        "PDCFilesystem": "com.liquidr.regulus.services.PDCFilesystem",
        "SV2PayloadManager": "com.liquidr.regulus.SV2payloads.SV2PayloadManager",
        "ScheduledThreadPoolExecutor": "java.util.concurrent.ScheduledThreadPoolExecutor",
        "VehicleID": "com.liquidr.datarep.VehicleID",
        "Wave Glider": "com.liquidr.regulus.Wave Glider",
        "Wiring": "com.liquidr.regulus.devices.Wiring",
        "airmarWaterspeed": "com.liquidr.regulus.devices.NmeaDriver",
        "airmarWeather": "com.liquidr.regulus.devices.AirmarWeatherDeviceDriver",
        "ais": "com.liquidr.regulus.devices.NmeaDriver",
        "aisService": "com.liquidr.regulus.services.ais.AISService",
        "amps": "com.liquidr.regulus.amps.AMPSManager",
        "bulkDataExport": "com.liquidr.regulus.services.BulkDataExportService",
        "canTransport": "com.liquidr.regulus.amps.CanTransport",
        "cellModem": "com.liquidr.regulus.devices.PoweredDevice",
        "commandReporter": "com.liquidr.regulus.wgmsinterop.CommandReporter",
        "commsManager": "com.liquidr.regulus.coms.CommunicationsManager",
        "config": "com.liquidr.regulus.config.Configuration",
        "configReporter": "com.liquidr.regulus.wgmsinterop.ConfigurationReporter",
        "debugClient": "com.liquidr.regulus.devices.PoweredDevice",
        "debugLAN": "com.liquidr.regulus.devices.PoweredDevice",
        "debugServer": "com.liquidr.regulus.devices.PoweredDevice",
        "deviceManager": "com.liquidr.regulus.devices.DeviceManager",
        "ethernetSwitch": "com.liquidr.regulus.devices.PoweredDevice",
        "ffn": "com.liquidr.regulus.nav.Navigator",
        "fileManager": "com.liquidr.regulus.wgmsinterop.FileManager",
        "fileSystemCleanup": "com.liquidr.regulus.services.FileSystemCleanupService",
        "floatAhrs": "com.liquidr.regulus.devices.FloatAhrs",
        "floatRecovery": "com.liquidr.regulus.devices.FloatRecoveryDevice",
        "gar": "com.liquidr.regulus.wgmsinterop.GenericAlarmReporter",
        "heading": "com.liquidr.regulus.services.HeadingService",
        "http": "com.liquidr.regulus.web.HttpService",
        "id": "com.liquidr.regulus.services.VehicleIDService",
        "internet": "com.liquidr.regulus.services.InternetAccessibilityService",
        "iridiumModem": "com.liquidr.regulus.devices.IridiumModem9603",
        "light": "com.liquidr.regulus.services.LightBar",
        "lightBar": "com.liquidr.regulus.devices.PoweredDevice",
        "linuxState": "com.liquidr.regulus.services.LinuxStateService",
        "linuxWatchdog": "com.liquidr.regulus.services.LinuxWatchdogService",
        "location": "com.liquidr.regulus.services.LocationService",
```

Liquid Robotics Proprietary and Confidential


```
        "log": "com.liquidr.regulus.logging.Log",
        "memoryMonitor": "com.liquidr.regulus.services.MemoryMonitor",
        "morse": "com.liquidr.regulus.services.TelegraphOffice",
        "nav": "com.liquidr.regulus.services.NavigationService",
        "otherSample": "com.liquidr.regulus.services.PersistantExternalService",
        "payloadPort1": "com.liquidr.regulus.SV2payloads.PayloadPort",
        "payloadPort2": "com.liquidr.regulus.SV2payloads.PayloadPort",
        "payloadPort3": "com.liquidr.regulus.SV2payloads.PayloadPort",
        "pdcParser": "com.liquidr.regulus.services.PDCParserService",
        "picture": "com.liquidr.regulus.services.PictureService",
        "pwrCtl": "com.liquidr.regulus.services.AMPSPowerWrapper",
        "radarReflector": "com.liquidr.regulus.devices.PoweredDevice",
        "rudder": "com.liquidr.regulus.devices.ReganRudder",
        "sample": "com.liquidr.regulus.services.PersistantExternalService",
        "scripting": "com.liquidr.regulus.services.ScriptingService",
        "serviceManager": "com.liquidr.regulus.services.ServiceManager",
        "setParametersListener": "com.liquidr.regulus.wgmsinterop.SetParametersListener",
        "setStatusHandler": "com.liquidr.regulus.wgmsinterop.SetStatusHandler",
        "sh": "com.liquidr.regulus.wgmsinterop.Runshell",
        "ship": "com.liquidr.regulus.nav.ShippingSituation",
        "smc": "com.liquidr.regulus.devices.PoweredDevice",
        "speedOverGround": "com.liquidr.regulus.services.SpeedOverGroundService",
        "telem06": "com.liquidr.regulus.wgmsinterop.Telemetry06Reporter",
        "thruster": "com.liquidr.regulus.devices.ReganThruster",
        "time": "com.liquidr.regulus.services.TimeService$Default",
        "torpedoGPS": "com.liquidr.regulus.devices.TorpedoGPS",
        "ublox": "com.liquidr.regulus.devices.UbloxDeviceDriver",
        "umbilical": "com.liquidr.regulus.devices.PoweredDevice",
        "uniqueIDManager": "com.liquidr.regulus.wgmsinterop.UniqueIDManager",
        "untwist": "com.liquidr.regulus.services.UntwistService",
        "uptime": "com.liquidr.regulus.services.UptimeService",
        "vehicleHealth": "com.liquidr.regulus.services.VehicleHealth",
        "virtualRelay": "com.liquidr.regulus.coms.VirtualRelayChannel",
        "wap": "com.liquidr.regulus.services.WirelessAccessPointService",
        "waterSpeed": "com.liquidr.regulus.services.WaterSpeedService",
        "weatherReporter": "com.liquidr.regulus.wgmsinterop.WeatherReporter",
        "weatherStation": "com.liquidr.regulus.services.WeatherStationService",
        "xp": "com.liquidr.regulus.SV3payloads.SV3PayloadManager",
        "xp0": "com.liquidr.regulus.SV3payloads.SV3PayloadDevice",
        "xp1": "com.liquidr.regulus.SV3payloads.SV3PayloadDevice",
        "xp2": "com.liquidr.regulus.SV3payloads.SV3PayloadDevice",
        "xp3": "com.liquidr.regulus.SV3payloads.SV3PayloadDevice",
        "xpSub": "com.liquidr.regulus.SV3payloads.SV3PayloadDevice"
    }
}
```

Appendix D. Towfish Battery Charger Technical Specifications

The LR VDC Towfish charger board is designed to power buffer sensor payloads towed behind a Wave Glider. It integrates two high voltage Inspired Energy Smart Batteries to provide uninterrupted battery power to an attached payload. By buffering input power to payloads, voltage droops are avoided during high current loads preventing sensor brownouts. Similarly, loads attached to the DC Towfish Charger do not need to be rated for the full voltage range of the Wave Glider's power rails.

The Towfish charger features two fixed input current limits – the board will draw less than 1 amp in SV2 mode and less than 2 amps in SV3 mode. This makes it compliant with SV2 Sensor Ports and with the newer SV3 Expansion Ports. Output voltage is entirely dependent on the attached Smart Battery. The Towfish Charger is smart charger compliant and capable of charging battery voltages as low as 16 volts DC and as high as 28 volts DC.

FEATURES:

- **Operating Voltage 9.0 V – 28 V**
- **Output Voltage battery-dependent**
- **Independent charge / discharge paths**
- **Dynamic power path control to prevent line droops**
- **Two Input current limits**
- **Compatible with Smart Batteries 14.4 – 33.6 VDC ***
- **Max continuous discharge current – 12 Amps**
- **Fully SMBus compatible**
- **Conforms to Smart Battery specification**
- **Designed to function with two Inspired Energy PH3059 Smart Batteries**

Liquid Robotics Proprietary and Confidential

Electrical Specifications

Figure D.1. Towfish Charger Block Diagram

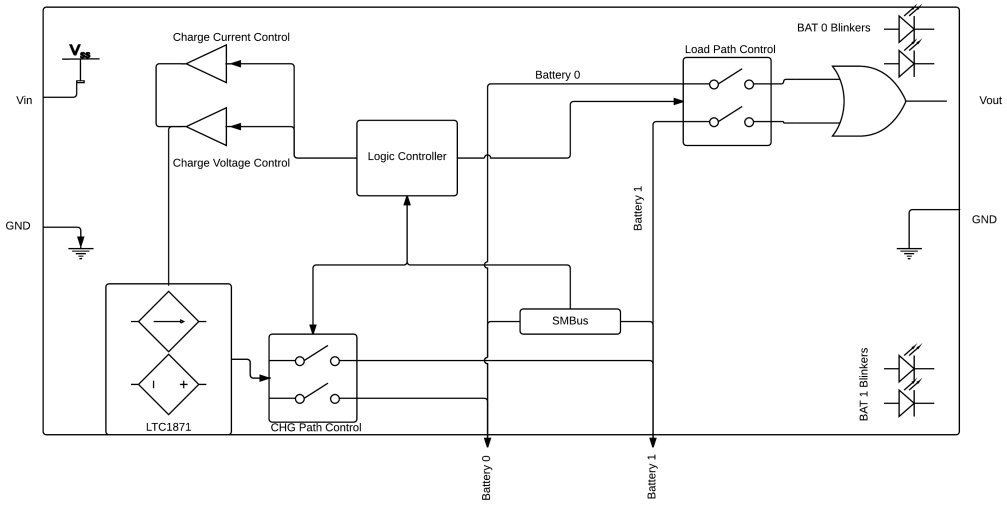


Table D.1. Absolute Maximum Ratings

Symbol	Parameter	Test Condition	Value	Unit
V_{in}	Charger input voltage		28	Volts
V_{out}	Output Voltage	Tested using the Inspired Energy 3059 (1)	33.6	Volts
I_{in}	Charger input Current	SV2 Mode	.3	Amps
		SV3 Mode	.8	
I_{out}	Output Current at terminals	Tested using the Inspired Energy 3059 (2)	12	Amps
R_{out}	Output resistance from battery connector to terminal		25	mOhms

(1) Vout is battery dependent.

(2) The rated continuous discharge of the PH3059 is 12 AMPS.

Liquid Robotics Proprietary and Confidential

Table D.2. Recommended Operating Conditions

Symbol	Parameter	Test Condition	Value	Unit
V_{in}	Charger input voltage	SV3 VCONS	16	Volts
V_{out}	Operating Temperature		25	Degrees Celsius

Connectors and Pinouts

Table D.3. Connector List

Reference Designator	Description
P1	Battery Voltage Out
P2	Ground Out
P3	Input Voltage
P4	Ground In
J2	Battery Interface 0
J3	Battery Interface 1
SW1	Current Mode Switch

Table D.4. Battery Interface Connector Pinout

Pin Number	Signal	Description
1	+ Battery	Positive Battery Terminal
2	+ Battery	Positive Battery Terminal
3	T	Battery Detect
4	- Battery / Ground	Negative Battery Terminal / Ground
5	+ Battery	Positive Battery Terminal
6	SCL	I2C Clock Signal
7	SDA	I2C Data Signal
8	- Battery / Ground	Negative Battery Terminal / Ground

Current Mode Switch:

The Current Mode Switch is used to change the input current limit of the charging device. Configuring the switch to SV3 Mode insures that the charger is compatible with an SV3's sub expansion port. Conversely, placing the charger into SV2 Mode drastically reduces input power and allows the device to be paired with typical SV2 sensor ports.

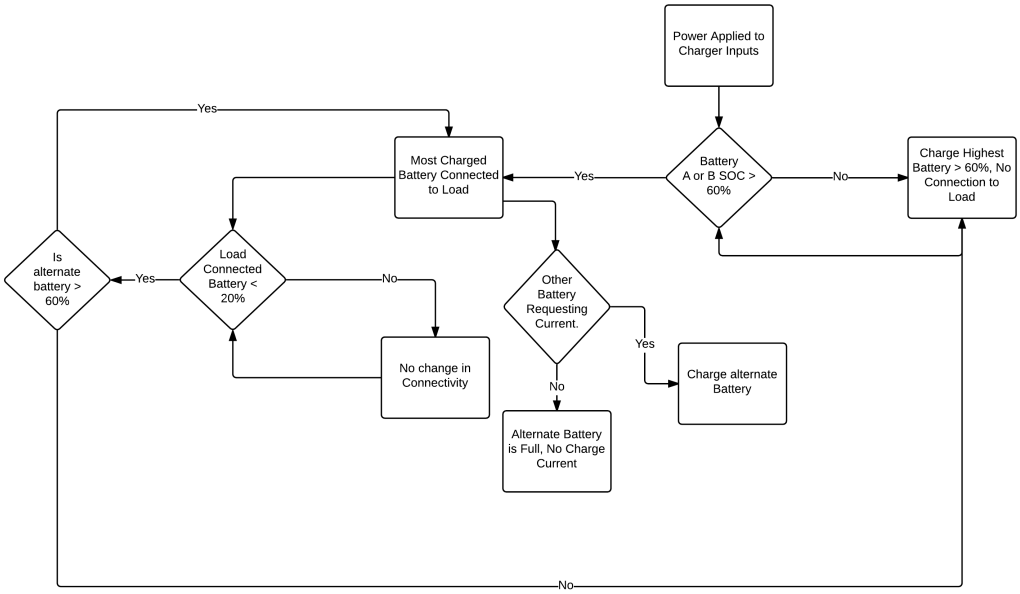
Liquid Robotics Proprietary and Confidential

Table D.5. DIP Switch Settings

DIP 1	DIP 2	Description
ON (1)	ON (1)	SV3 Mode
OFF (0)	OFF(0)	SV2 Mode
ON (1)	OFF (0)	Reserved
OFF (0)	ON (1)	Reserved

Functional Description

Figure D.2. Functional Flow Chart



Charger behavior is dictated by the current state of charge of the attached batteries.

Table D.6. State of Charge Thresholds

Threshold	Value	Description
State of Charge (SOC)	> 60%	A battery with SOC > 60% is eligible to provide power to the output load
	< 20%	Once a battery is < 20% charged, it is considered discharged and disconnected from the output load

The Tow Body Charger has two sets of visual indicators to represent both connected batteries. These indicators are located on the output side of the PCB providing visual clues to the behavior of the charger with respect to Battery 1 and Battery 2.

Liquid Robotics Proprietary and Confidential

Table D.7. LED Interface Indicators

LED	Description
Solid Red	Battery is disconnected from both the charger and the load
Blinking Red	Battery is connected to the charging circuit and actively charging
Blinking Green	Battery is connected to the load and actively discharging.
Solid Green	Reserved
Blinking Green, Blinking Red	Reserved

Device Power Up

On power up, no initial voltage is applied to the output pins. At this point the controller determines whether one, or both batteries are present, and their current state of charge.

Depending on the condition of the batteries, the charger may opt to connect a battery to the output, or to the charging circuit. At no point is a battery connected to both the charging circuit and the output.

If neither battery is sufficiently charged enough to provide power to the output, it is charged until it crosses the acceptable charge threshold. If both batteries are above this threshold the battery with the highest state of charge is connected to the output.

Charging

Charging is handled by periodically polling the smart battery for its optimum voltage and current profile required for charging. At no point is a battery connected to both the load and the charger.

Similarly, once the battery requests either null current, or the “Fully Charged” flag is raised over the SMBus, the charger is disconnected from the respective battery.

Wakeup Current

If a battery is not responding on the SMBus, it is assumed to be critically discharged. The Tow Body Charger complies with the Smart Battery standard and will attempt to provide a wakeup current to revitalize the discharged battery, if the battery is physically detected (using the T pin on Smart Batteries). The charger will attempt to do this a maximum of two times. Power cycling the charger will reset the number of times it will attempt to initialize a dead battery.

Power Provided To Loads

Whether or not the loads receive power is dependent on the state of charge of the overall system. If any of the attached batteries are above 60% charged, they are eligible to power the output load. In all circumstances, the most charged battery will be attached to the output. The most charged battery will

Liquid Robotics Proprietary and Confidential

remain connected to the load until it is discharged below the minimum state of charge – 20% charged. At this point, if the secondary battery is present and above the acceptable charge level, the charger transitions the load to the secondary battery.

It is possible (and common) for neither battery to power the load. The charger will not attach a battery to the output if it is less than 60% charged. This condition is often present after systems have remained off for extended periods of time, or on first boot from factory. In this case, the charger will charge one of the two batteries above the acceptable charge limit, before attaching it to the load.

Consequently, it is possible to place the charger in a "charge – discharge" cycle where the load is temporarily powered off. This situation occurs when the load power draw average is significantly higher than the charger's ability to recharge the buffer batteries. Careful consideration of the load, battery capacity, and load duty cycle should be taken into account when calculating load up-time.

Operation with a Single Battery

It is possible to operate the Towfish Charger with a single Smart Battery. The charger will continue to obey its state of charge laws. If the attached battery has greater than acceptable state of charge (60%), it will attach the battery to the load. Similarly, if the state of charge dips below the minimum state of charge (20%), the charger will disconnect the battery from the load and begin charging the battery.

Without a second battery, the charger periodically removes power from the loads to recharge the single buffer battery. As a result, the uptime of the output is impacted drastically. **Operating with a single Smart Battery is not recommended.**

Damaged Attached Smart Battery

The Towfish Charger is capable of detecting damaged Smart Batteries. Certain circumstance can cause a battery to malfunction – temperature, cable damage etc. If these conditions are detected, the system will operate exclusively on the remaining working battery. If neither Smart Battery is functional, the charger will terminate and disconnect all current paths (charger and load).

Calculating Power and Load Requirements

The battery charger limits input current. Power into the charger can be roughly categorized as:

$$Pin = .8 * VCONS$$
$$Assuming\ VCONS = 9\ VDC\ (worse\ case)$$
$$Pin = 7.2\ W$$
$$Pout = Output\ Power$$
$$Pout = Iout * Vbat$$
$$Dcycle = (Time\ On)/(Time\ on + Time\ Off)$$

Applications can be broken down into two different categories:

Operating at a Power Surplus – If Pout of load is less than chargers power input the unit will operate without interruption. A switch over will happen every:

Liquid Robotics Proprietary and Confidential

$$TSwitch = (Capacity\ W * H) * .8/(Pout * Dcycle)$$

Operating at a Power Deficit – Operating at a power deficit will force the charger into an "On-Off – Charge" cycle. Assuming both attached batteries are fully charged to start:

The first Switch occurs at:

$$TSwitch = (Capacity\ W * H) * .8/(Pout * Dcycle)$$

The second Switch occurs at:

$$TSwitch2 = (Capacity\ W * H) * .8/(Pout * Dcycle)$$

If

$$TSwitch2 > Capacity * \frac{.6}{Pin}$$

The charger will be able to maintain continuous power. Otherwise, the charger will be able to sustain an ON / OFF cycle with a percentage time ON of:

$$\frac{\frac{.6}{Pout * Dcycle}}{\left(\frac{.6}{Pout * Dcycle} + \frac{.6}{Pin}\right)}$$

Appendix E. Relay Header Data

Iridium Vehicle-to-Remote Applications Data Packets

Vehicle-to-remote applications relay packets are command responses, telemetry or alarms. These packets have a 51 byte header, followed by up to 205 bytes of response data.

Iridium Vehicle-to-Remote Applications Field Name	Byte #	Length (bytes)	Value	Format
Protocol Revision	0	1	1	Big Endian
Message Length	1-2	2	Total length - 3	
MO_Header IEI	3	1		
MO_Header Length	4-5	2		
CDR Reference	6-9	4	“Msg1”	
IMEI	10-24	15		
Session Status	25	1		
MOMSN	26-27	2		
MTMSN	28-29	2		
Time of Session	30-33	4		
MO_Payload IEI	34	1		
MO Payload Length	35-36	2	[# Level 2 bytes]	
MO Location Info IEI	37	1		
MO Location Info Length	38-39	2		
MO Lat/Long	40-46	7		
CEP Radius	47-50	4		
Level 2 Packet (Response or Telemetry)	51-255	≤ 205		Little Endian

Iridium Vehicle-to-Remote Applications Field Name Descriptions:

Protocol Revision – Iridium protocol revision number.

Message Length – Number of bytes in the remainder of the packet.

MO_Header IEI – Mobile Originated Header Information Element Identifier.

MO_Header Length – Mobile Originated Header length in bytes.

Liquid Robotics Proprietary and Confidential

CDR Reference – Call Detail Record Reference.

IMEI – International Mobile Receiving Identity. 15 digit code of the receiving modem.

Session Status – Status of the connection link. Values larger than 2 indicate a problem.

MOMSN – Mobile Originated Message Sequence Number.

MTMSN – Mobile Terminated Message Sequence Number.

Time of Session – Time of day of session.

MO_Payload IEI – Mobile Originated Payload Information Element Identifier.

MO Payload Length – Mobile Originated Payload Length. (Number of Level 2 bytes)

MO Location Info IEI – Mobile Originated Location Information Element Identifier.

MO Location Info Length – Mobile Originated Location Information Length.

MO Lat/Long – Mobile Originated Latitude and Longitude data. See below.

CEP Radius – Circular Error Probability Radius. Distance (in Kilometers) around the center point within which the IMEI is located with an 80% probability of accuracy.

Level 2 Packet – Telemetry or Command response (see Chapter 6: *Vehicle Responses* and Chapter 7: *Vehicle Telemetry*)

Mobile Originated Lat/Long Data

The Mobile Originated Latitude and Longitude data is encoded in a 7-byte value as follows:

Field Name	Bytes	Units	Value Description
Latitude N-S Bit Longitude E-W Bit	1	N/S/E/W	Bit 6: 0 → North, 1 → South Bit 7: 0 → East, 1 → West
Latitude (degrees)	1	Degrees	[0 to 90] (dec) [0x00 to 0x5A] (hex)
Latitude (minutes)	2	Minutes / 1,000	[0 to 59,999] (dec) [0x0000 to 0xEA5F] (hex)
Longitude (degrees)	1	Degrees	[0 to 180] (dec) [0x00 to 0xB4] (hex)
Longitude (minutes)	2	Minutes / 1,000	[0 to 59,999] (dec) [0x0000 to 0xEA5F] (hex)

Relay Vehicle-to-Remote Applications Packets

Vehicle-to-Remote Applications Relay packets are similar (but not identical) to the Iridium Vehicle-to-Remote Applications packets. These packets also have a 51 byte header, followed by up to 205 bytes of response data.

Though these packets are structured like Iridium packets, note that an ID number and IP Address/Port replace the IMEI. Also, many of the fields are zero-valued placeholders.

Relay Vehicle-to-Remote Applications Field Name	Byte #	Length (bytes)	Value
Protocol Version	0	1	See below
Message Length	1-2	2	Total length - 3
MO Header IEI	3	1	0
MO Header Length	4-5	2	0
CDR Reference	6-9	4	See below
Vehicle Unique ID	10-13	4	See "Vehicle Unique ID" (page 84)
IP Address	14-17	4	
IP Port	18-21	4	
(not used)	22-24	3	0
Session Status	25	1	0
MOMSN	26-27	2	0
MTMSN	28-29	2	0
Time of Session	30-33	4	0
MO Payload IEI	34	1	0
MO Payload Length	35-36	2	[# Level 2 bytes]
MO Location Info IEI	37	1	0
MO Location Info Length	38-39	2	0
MO Lat/Long	40-46	7	0
CEP Radius	47-50	4	0
Level 2 Packet	51-255	≤ 205	

The Relay Packet significant field types are:

Protocol Version – Number indicates the type of protocol to be expected in the Level 2 packet.

Protocol	Value	Notes
Iridium (plug-In)	1	[1]
XBee **(SV2 Only)**	2	
Serial	3	
802.11	4	
Alternate Iridium	5	[2]

Notes:

[1] Protocol 1 – Refers to the Iridium modem in the CCU. Uses SBD transmit/receive.

[2] Protocol 5 – Refers to another device on the vehicle that sends and receives data in the Iridium format. This data transmission does not use SBD.

CDR Reference – A 4-character text sequence that indicates the type of message recipient, as follows:

Liquid Robotics Proprietary and Confidential

Protocol	Value
Iridium (plug-In)	“Msg1”
XBee **(SV2 Only)**	“Xbee”
Serial	“Ser1”
802.11	“WiFi”
Alternate Iridium	“Irid”

Vehicle Unique ID – The unique 4-byte identification number that is stored in the vehicle's Non-Volatile memory. See “Vehicle Unique ID” (page 84).

IP Address – Internet Protocol destination address of the message recipient.

IP Port – Internet Protocol port number of the destination address.

MO Payload Length – Length of the Level 2 Content payload.

Level 2 Content – Data in the format specified by the "Protocol Version" number.

Virtual Relay Vehicle-to-Remote Applications Packets

These packets usually contain telemetry data. Related Ack and Nak packets (up to 270 bytes long) have up to 340 bytes of payload and 50 bytes of header content.

Field Name	Index	Length (bytes)	Value
Protocol Revision	0	1	See: Protocol Revision
Message Length	1	2	Total length - 3
MO Header IEI	3	1	0
MO Header Length	4	2	0
CDR Reference	6	4	See: CDR Reference
Unique Id	10	4	
Internal IP Addr	14	4	
Session Id	18	4	
Length	22	2	0-Unlimited, 1 -64K
Spare	24	1	0
Session Status	25	1	0
MOMSN	26	2	0
MTMSN	28	2	0
Time Of Session	30	4	Unix Timestamp format
MO Location Info IEI	34	1	0
MO Location Info Length	35	2	0
MO Lat/Long	37	7	0
CEP Radius	44	4	0

Liquid Robotics Proprietary and Confidential

MO Payload IEI	48	1	0
MO Payload Length	49	2	***
Interface Level 2 Content	51	Up to 339	

*** For virtual relay status commands, the MO Payload Length value is the total buffer size in bytes used by the virtual relay client to send multiple commands. The buffer allows for multiple TCP requests for incomplete transmission.

Iridium Remote Applications-to-Vehicle Data Packets

A Remote Applications-to-Vehicle Iridium packet carries commands to the vehicle. It has 30 bytes of header content and can carry up to 135 bytes of command data.

Iridium Remote Applications-to-Vehicle Field Name	Byte #	Length (bytes)	Default Value	Format
Protocol Revision	0	1	1	Big Endian
Message Length	1-2	2	Total length – 3	
MT_Header IEI	3	1	65	
MT_Header Length	4-5	2	21	
CDR Reference	6-9	4	"Msg1"	
IMEI	10-24	15		
MT_Disposition Flags	25-26	2	0	
MT_Payload IEI	27	1	66	
MT_Payload Length	28-29	2	[# Level 2 bytes]	
Level 2 Command Packet	30-164	≤ 135		Little Endian

Iridium Remote Applications-to-Vehicle Field Name Descriptions:

Protocol Revision – Iridium protocol revision number.

Message Length – Number of bytes in the remainder of the packet.

MT_Header IEI – Mobile Terminated Header Information Element Identifier.

MT_Header Length – Mobile Terminated Header length in bytes.

CDR Reference – Call Detail Record Reference.

IMEI – International Mobile Receiving Identity. 15 digit code of the receiving modem.

MT_Disposition Flags – Mobile Terminated Disposition Flags.

Liquid Robotics Proprietary and Confidential

MT_Payload IEI – Mobile Terminated Payload Information Element Identifier.

MT_Payload Length – Number of bytes in the payload packet that follows.

Level 2 Packet – Command Data. (see Chapter 5: *Commands*)

Relay Remote Applications-to-Vehicle Packets

A Remote Applications-to-Vehicle Relay packet carries commands to the vehicle. It has 30 bytes of header content and can carry up to 135 bytes of command data.

Iridium Remote Applications-to-Vehicle Field Name	Byte #	Length (bytes)	Default Value
Protocol Version	0	1	See below
Message Length	1-2	2	Total length – 3
MT Header IEI	3	1	65
MT Header Length	4-5	2	21
CDR Reference	6-9	4	See below
Vehicle Unique ID	10-13	4	See "Vehicle Unique ID" (page 84)
IP Address	14-17	4	
IP Port	18-21	4	
(not used)	22-24	3	0
MT Disposition Flags	25-26	2	0
MT Payload IEI	27	1	66
MT Payload Length	28-29	2	[# Level 2 bytes]
Level 2 Command Packet	30-164	≤ 135	

Relay Remote Applications-to-Vehicle Field Name Descriptions:

The **Protocol Version** and **CDR Reference** numbers are the same as used for Vehicle-to-Remote Applications Relay packets.

Protocol Version	Value
Iridium (plug-in)	1
XBee **(SV2 Only)**	2
Serial	3
802.11	4
Iridium (alternate)	5

CDR Reference	Value
Iridium (plug-in)	"Msg1"

Liquid Robotics Proprietary and Confidential

CDR Reference	Value
Xbee **(SV2 Only)**	"Xbee"
Serial	"Ser1"
802.11	"WiFi"
Iridium (alternate)	"Irid"

Vehicle Unique ID – The unique 4-byte identification number that is stored in the vehicle's Non-Volatile memory - see “Vehicle Unique ID” (page 84).

IP Address – Internet Protocol address of the command originator.

IP Port – Internet Protocol port number of the command originator.

Level 2 Packet – Data in the format specified by the "Protocol Version" number.

Virtual Relay Remote Applications-to-Vehicle Packets

Single Virtual Relay Remote Applications-to-Vehicle packets can be up to 370 bytes long, with up to 320 bytes of payload data and 30 bytes of Relay defined header content. This type of packet is used to send commands from remote applications to vehicles.

Field Name	Index	Length (bytes)	Default Value
Protocol Revision	0	1	See: Protocol Revision
Message Length	1	2	Total length - 3
MT Header IEI	3	1	65
MT Header Length	4	2	21
CDR Reference	6	4	See: CDR Reference
Unique Id	10	4	
Internal IP Addr	14	4	
Session Id	18	4	
Spare	22	3	0
MT Disposition Flags	25	2	0
MT Payload IEI	27	1	66
MT Payload Length	28	2	Payload length
Interface level 2 Content	30	Up to 270	

Appendix F. CRC-16 Calculation

A packet's error-check field contains a 16-bit CRC calculated from the bytes in a data packet. The specific bytes used for the calculation will depend on the type of packet. However, the computational algorithm will be the same.

Here is an example of the algorithm written in C#:

```
public static CRC16 ComputeFromArray(byte[] buffer, uint offset, uint count, UInt16 seed)
{
    UInt16 crc = seed;

  for (uint idx = offset; count-- > 0; ++idx)
    crc = ComputeIncrementally(crc, buffer[idx]);

  return new CRC16(crc);
}

public static UInt16 ComputeIncrementally(UInt16 crc, byte a)
{
  int i;

  crc ^= a;

  for (i = 0; i < 8; i++)
  {
    if ((crc & 1) == 1)
    {
       crc = (UInt16)((crc >> 1) ^ 0xA001);
    }
    else
    {
       crc = (UInt16)(crc >> 1);
    }
  }

  return crc;
}
```

Appendix G. PIB and PM Slot Configuration Commands for SV2 Payloads Adapted to SV3 Vehicles

Overview

The Liquid Robotics 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 Liquid Robotics sensor-specific applications run on PIBs providing a management and communications interface between WGMS and sensors that Liquid Robotics 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 Liquid Robotics sensor-specific applications running on the SMC Linux operating system.

The following sections provide information on the commands used to configure the PIB and its PM slots - they do not provide information on Liquid Robotics sensor-specific commands for managing Liquid Robotics integrated sensor payloads. If you have a legacy Wave Glider SV2 vehicle with an Liquid Robotics integrated sensor, see the *Wave Glider® Integrated Sensor User Guide* for that sensor to obtain information on sensor-specific commands that run on a PIB or on a SMC.

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 “Empty ID” (page 339).
- The COMM program configures a PM slot to allow a connected device to communicate using the Liquid Robotics 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.

Liquid Robotics Proprietary and Confidential

- 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 CCU 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 "PIB ID" (page 338). 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.

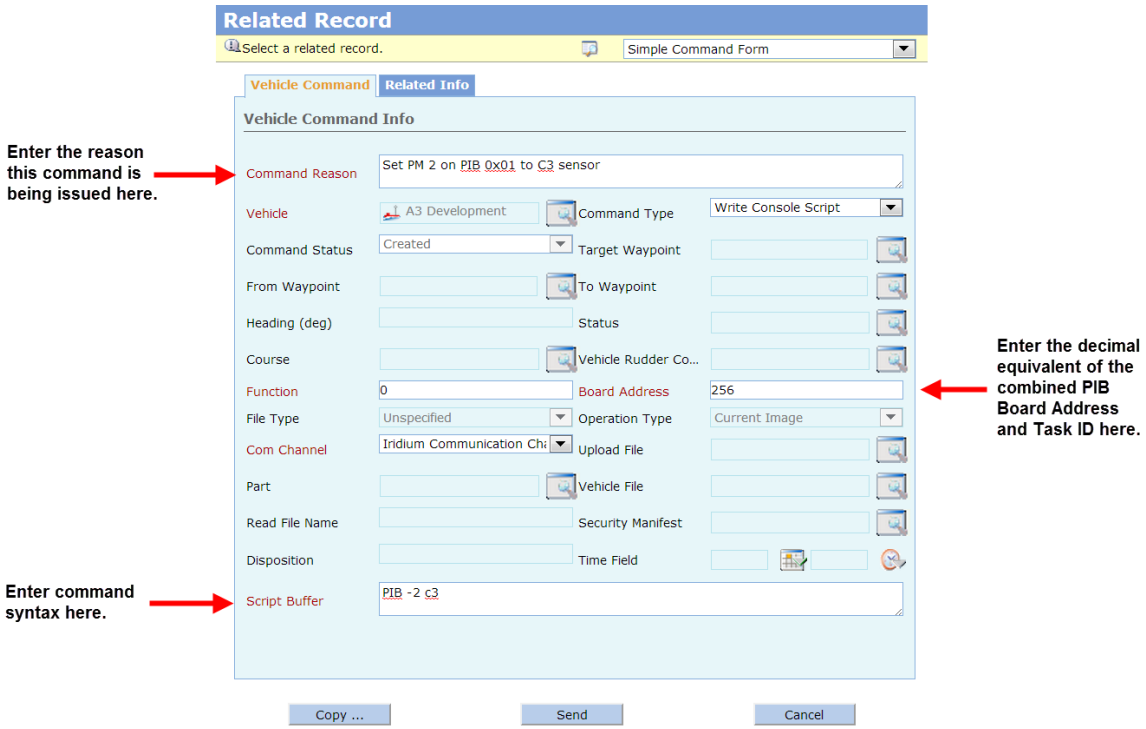
   If the Board ID for the PIB is `0x02`, and the command is directed to PM slot `03`, the combined hexidecimal values would be `0x0203`. To determine the decimal version, multiply the Board ID value by 256 and add the Task ID {(02)(256) + 03}. The result is 515. This is the number you would enter into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

Liquid Robotics Proprietary and Confidential

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 "Empty ID" (page 339).

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*]

Liquid Robotics Proprietary and Confidential

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 CCU 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 CCU 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.

Liquid Robotics Proprietary and Confidential

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 Liquid Robotics 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 "PIB PM Slot Configuration Commands" (page 336) 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

Liquid Robotics Proprietary and Confidential

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.