

↑	Software Release Note for Sensor Management Computer (SMC) Core Components version 2.2.0 			
	responsible engineer	M. Cookson	created	8/6/2014
	product name	SMC Core Components	last updated	8/6/2014
	software version	2.3.0	Status	Final
	software part number	02617-28		
	document part number/rev	02619-28 Rev A		

TABLE OF CONTENTS

Overview.....	4
Changes in Version 2.3.0 (08/01/2014, P/N 02617-28).....	5
New Features.....	5
Bug Fixes	7
Dependencies	9
Open Issues.....	10
End-user Impacts	10
Support Impacts	10
Changes in Version 2.2.0 (05/30/2014, not released)	10
New Features.....	10
The result is the ack code followed by the response. For example:	11
Bug Fixes	11
Dependencies	12
Open Issues.....	12
End-user Impacts	12
Support Impacts	12
Changes in Version 2.1.0 (04/30/2014, P/N 02617-26).....	12
New Features.....	13
Bug Fixes	20
Dependencies	22
Open Issues.....	22
End-user Impacts	23
Support Impacts	23
Changes in Version 2.0.1 (02/07/2014, P/N 02617-25).....	23
New Features.....	23
Bug Fixes	25
Dependencies	27
Open Issues.....	27
End-user Impacts	27

Support Impacts	27
Installation/Upgrade Instructions	28
Setting up the SD Card	30
Working with the SMC and the SV2 Float C&C	30
Working with the SMC and the SV3 via Sensor Port	32
Working with the SMC and the SV3 via Expansion Port	33
Installing the Application	36
Changing Encryption Passphrases	38
Component Configuration	39
Aquatracka (Chelsea Technologies Group) Configuration	41
Commands	41
BenthosModem Configuration	43
Configuring the Client Program	43
Loading the Modem Firmware	43
Configuring Modem Parameters	44
C3 (Turner Designs) Configuration	45
Commands	45
Events and Telemetry	45
Common XML file keys to edit	47
ConnectionManager Configuration	47
Common XML file keys to edit	47
DataRecorder Configuration	48
Common XML file keys to edit	48
Using the DataRecorder	49
GPSListener Configuration	50
Commands	50
CORBA Events	51
XML Configuration	52
Logger	54
GPSWaves Configuration (GPS and IMU based waves sensor)	54
Commands	55
CORBA Events	55
XML Configuration	56
LuaWrapper Configuration	57
Commands	57
Special Variables	58
CORBA Event Tables and Variable Definitions	60
Pre-defined Callback Functions	62

XML Configuration	63
Sample lua script.....	63
Magnetometer Configuration (Marine Magnetics Explorer)	65
Commands	65
CORBA Events.....	67
XML Configuration	68
MAM Configuration (Datawell MOSE).....	68
Commands	68
Common XML file keys to edit	70
PayloadInterface Configuration.....	70
PayloadInterface Commands.....	71
PPPHost Configuration.....	75
Common XML file keys to edit	75
Configuring the RUDICS Modem	75
Configure the dialup script	76
Configure the PPP connection script	76
Manual connection to Iridium network	76
Setting up a key file for scp/ssh (dbclient) connection without password	79
SeabirdGPCTD Configuration.....	80
SeabirdGPCTD Telemetry Output	82
SeabirdGPCTD Commands	84
SimpleSBDQueue Configuration.....	85
Commands	85
CORBA Events.....	86
XML Configuration	87
timelimit2.....	87
VirtualRelay Configuration	88
Common XML file keys to edit	88
Typical Virtual Relay configurations.....	90
Commands	90
VemcoVR2C Configuration	91
Common XML file keys to edit	91
Using the VemcoVR2C.....	92
WetLabsSensorSimulator.....	92
Appendix 1 – Sample Application Configuration File	93
Appendix 2 – Sample Telemetry Configuration	96
History.....	111

Overview

The Sensor Management Computer (SMC) core components encompass the superset of standard applications, libraries, scripts and configuration files that might be loaded onto the embedded SMC Linux computer. These include:

COMPONENT	DESCRIPTION	VERSION
Aquatracka	Interface to Chelsea Aquatracka fluorimeter	1.1.8+
BenthosModem	Interface to Benthos acoustic modems (tested with version 8.2.2)	1.0.10+
busy	A utility that can be invoked from script to keep the CPU from going idle (busy on) or to turn that mode off (busy off)	1.1.8+
C3	Interface to Turner Designs C3 fluorometer using command mode	1.3.0+
catgps	This is a utility that can be used during board setup or troubleshooting to configure and show the output of the GPS module when the SMC software is not running.	1.1.8+
ConnectionManager	Listens to requests for and manages TCP/IP connections, such as a cell modem	1.1.9+
ConfigOctalUartLine	Configures an octal UART port electrical protocol (must be executed as root)	1.0.9+
ConfigurePower	Configures the power-saving features of the platform (must be executed as root)	1.0.9+
DataRecorder	A module capable of archiving received serial port data and making it available for extraction at runtime	1.1.0+
DemoSensor	An example sensor implementation	1.0.9+
EnableEthernet	Activates the Ethernet interface and establishes an IP address via DHCP	1.0.9+
EventManager	CORBA-based event subscription and distribution service	1.0.9+
GPSTListener	Interface to GPS receiver	1.0.9+
GPSWaves	Wave height sensor based on GPS or IMU input	2.0.1+
InitOctalUartLines	Configures the electrical protocol for all octal UART ports (must be executed as root)	1.0.9+
InitPowerSwitches	Configures the power switches to their initial (off) state (must be executed as root)	1.0.9+
killsmc	Stops all SMC processes (must be executed as root)	1.0.9+
LISST	Interface to Sequoia Instruments' LISST sensor	1.7.0+
Logger	A utility program that accepts text from standard input and writes to a log file set with configurable maximum size	1.1.0+
LogListener	Listener for reported log events from the Float C&C	1.1.2+
LuaWrapper	Wrapper for a lua interpreter instance: allows execution of lua scripts	1.1.2+
Magnetometer	Interface to Marine Magnetics' Explorer Magnetometer	1.7.0+
MakeThumbnail	Creates a reduced resolution image of a picture.	1.1.8+
MakeZoom	Creates a cropped image from a picture.	1.1.8+
MAM	Interface to Datawell motion sensor	1.0.9+
MicrostrainTest	Test interface to Microstrain GDM-GX35 GPS and IMU sensor	2.0.1+
MobileCtrl	Mobile WGMS controller, which enables local piloting through a radio network (<i>alpha version, not documented</i>)	2.0.1+
omniNames	CORBA (omniORB) name server	1.0.9+
PayloadInterface	Implementation of Float C&C payload protocol, which exposes services registered via CORBA to the Float C&C and WGMS	1.0.9+
PayloadManager	A CORBA names registry that can be enumerated	1.0.9+
PPPHost	Manager of a point-to-point protocol connection (typically RUDICS) and executor of configured scripts while the internet connection is open	1.0.9+
ReportBoardVersion	Reports the version number of the SMC board ("ReportBoardVersion" or "ReportBoardVersion -v")	1.1.8+
SeabirdGPCTD	Interfaces a Seabird GPCTD (Glider Payload CTD)	2.3.0+
ServerExec	A utility that contacts an application via its CORBA-	1.0.9+

COMPONENT	DESCRIPTION	VERSION
	registered name and invokes a command, returning any output.	
SetPowerSwitch	Turns one of the SMC power switches on or off (must be executed as root)	1.0.9+
SimpleSBDQueue	Interface to an SMC-local 9522B/9523 Iridum SBD modem	1.7.0+
TakePicture	Powers on an attached camera, sets zoom level, snaps a picture and downloads the image.	1.1.8+
timelimit2	Utility that limits runtime to a maximum time	1.1.2+
VemcoVR2C	Interface to the Vemco VR2C sensor	1.1.3+
VirtualRelay	Uses a TCP/IP connection to send and receive messages from WGMS	1.1.9+
WetLabsSensor	Interface to Wet Labs sensors like Eco Puck and CStar	1.0.10+
WetLabsSensorSimulator	Generate simulated Eco Puck samples over a pseudo serial port.	1.1.0+
WHOIMM	Interface to a WHOI Micro Modem	1.2.0+

Note that this distribution is intended for those who are comfortable with Linux and making configuration changes. Other installation kits for specific configurations are also available.

Changes in Version 2.3.0 (08/01/2014, P/N 02617-28)

This release completes the requested Marine Magnetics Magnetometer feature set and introduces support for the Seabird GPCTD on the SMC. It also fixes a number of bugs.

New Features

SWSMAC-472 "Support new ack with text explanation for console acks"

Description

Implement NakWithDescription and use it to respond to Write Console Ack. Any output from the command is carried in the body of the message as with Console Ack. Verify with both SV2 and SV3. Success information should be added to the CORBA SendCommandComplete message: services and scripts may be updated incrementally to supply this information.

The SV3 Expansion Port will use the information to flag a command as successful or unsuccessful, according to the SV3 REST interface protocol. Additional changes may be required to Regulus to interpret the result as a NakWithDescription response.

Resolution

For SV2-style interfaces, PayloadInterface now sends NAK_WITH_DESCRIPTION (type ID 0x27) instead of CONSOLE_ACK in the following cases:

1. When catching an exception while calling the SendCommand method of CORBA server
2. When the new "Successful" flag in the "Command Complete" event is false (by default it is true for backward compatibility.)

For SV3-style interfaces, PayloadInterface passes the command result back by setting the "success" parameter to false in the above situations. Regulus will need to change the message sent back to WGMS in these cases.

Programmers on the SMC can participate in the following ways:

1. If you throw an exception (other than WillCompleteAsynchronously) from the SendCommand CORBA method, PayloadInterface will send the Nak With Description instead of Console Ack. So all of the cases where bad parameters were passed should work automatically.

2. If your program sends the SendCommandComplete event, you can call the new Successful() method and pass false if the command should be NAK'd.
3. If your program uses SensorProgram, SensorController and SensorConcreteCmd, throwing an exception from the command class will cause the command to be NAK'd. You can also call the new SetSuccessFlag() member of the command class and pass false to indicate that the command failed.
4. From a lua script, call the new SetSuccessFlag() member of the SMC variable and pass false to indicate that the script failed.

```
function Start2()  
    gasynch = SMContext  
    gasynch:SetAsynch()  
    gasynch:AppendOutput("ERROR")  
    gasynch:SetSuccessFlag(false)  
    gasynch:ReportComplete()  
end
```

SWSMAC-474 "GPSWaves: update GPSWavesLib to return average and peak periods and spectrum data dir & psd"

Description

Use the updated UnifiedWavesLib from Jim Thomson (ver: June2014) to return average & peak periods in addition to existing wave height report parameters. The output spectrum data now also includes spectral direction & directional spread.

Resolution

The waves report now includes average & peak period Ta & Tp along with significant wave height and direction.

Also, the spectrum output now also contains spectral direction & directional spread.

SWSMAC-475 "Create Seabird GPCTD sample interface for SMC"

Description

Create an SMC interface to the Seabird GPCTD as a well-documented sample source project for developers.

Resolution

A new project "SeabirdGPCTD" was created and new runtime configurations were added with an accompanying XML file for configuration. The sensor sends back samples via telemetry and optionally logs low-level messages and samples to a configurable log set.

The supported commands are:

Command:

```
ctd --expert <command>
```

Description:

Send one of the supported device commands and get the output from it. The supported commands are:

```
dc (display calibration coefficients)  
ds (display status)  
getcc (get calibration coefficients)  
getcd (get configuration data: XML)  
getec (get event counters: XML)  
gethd (get hardware configuration: XML)  
getsd (get status data: XML)  
sl (send last sample)  
slp (send last pressure sample)
```

Command:

```
ctd --autorun <on-or-off>
```

Description:

Configures the sensor interface to begin sampling (on/true) when the service is started or not to do so (off/false)

Command:

```
ctd --log-raw <on-or-off>
```

Description:

Enables (on/true) or disabled (off/false) logging of all messages and responses from the sensor.

Command:

```
ctd --power <on-or-off>
```

Description:

Enables (on/true) or disabled (off/false) power to the sensor. Note that power is automatically turned on when sampling is started, so there's no need to send an additional command to power it on.

Command:

```
ctd --start { <period(sec)> { <samples-per-block> { <flush-time(sec)> { <off-time(sec)> } } } }
```

Description:

Turns on power to the sensor and starts taking readings, returning telemetry for each sample. If included, the parameters overwrite the currently-configured values (the new values are saved persistently.)

- period(sec) is the number of seconds between samples. Note that this value cannot be over 14 seconds if an O2 sensor is installed (see GPCTD manual for further detail.)
- samples-per-block is the number of samples per telemetry sample, and number of samples between imposing an off time.
- flush-time(sec) is the number of seconds to run the pump (to get representative material in the line) before starting to sample.
- off-time(sec) is the number of seconds to suspend pumping and sampling between blocks. If this value is zero, sampling is continuous.

Command:

```
ctd --status
```

Description:

Returns a message representing the state of the sensor.

Command:

```
ctd --stop
```

Description:

Stops the sampling loop, but leaves the sensor powered (in low power state.)

Bug Fixes

SWSMAC-8 "Compilation note "note: the mangling of 'va_list' has changed in GCC 4.4" in build"

Description

The use of va_list now generates warnings with the GCC version 4.4.4 It would be better if the messages did not show up since there's not much that can be done about it. The

problem has been documented here: "http://gcc.gnu.org/bugzilla/show_bug.cgi?id=42748"
There may be a patch that can be applied to clear this up.

Resolution

-Wno-psabi was added to Zoom builds; this turns off notifications.

SWSMAC-469 "Magnetometer Report Events to Shore"

Description

There are some events that should be reported to shore intelligently:

1. Background clock reset
2. Automatic clock resync. Magnetometer clock is out of sync with the onboard clock
3. Magnetometer reports an "Instrument Mistuned" or 'M' warning and retunes; this results in data loss
4. Null character and/or garbage data
5. Magnetometer has not reported data in M minutes. Was there power loss or was it disconnected? (not implemented)
6. Magnetometer was hot swapped; that is the Magnetometer startup sequence was detected. This is expected when Magnetometers are hot swapped, but what if this occurs in an unexpected situation?

Resolution

All items except #5 were implemented.

A new AlarmSubsystem class in PayloadInterface handles an XML-defined CORBA message (by default, "AlarmMessage"), which carries a format defined in EventLib's AlarmMessage class. When that message is received, it sends System Information¹ message (type ID 0x71) event record v2 (subtype 5) using the fields in the event. Note that different kinds of alarms can be sent:

- one-time event messages
- one-time alarm messages
- persistent alarm messages (asserting or clearing the alarm)

Event messages have a free-form text field and an enumerated type field. PayloadInterface supports this asynchronous message type for both SV2 and SV3 style payloads.

PayloadInterface also implements a new command as follows to announce events from ServerExec:

Alarm Type	Command
0	SendAlarmCleared sender=<address> text=<text> {reporter=<ID>} {notification=<ID>} {enum=<ID>} {data=<hex string>}
1	SendAlarmSet sender=<address> text=<text> {reporter=<ID>} {notification=<ID>} {enum=<ID>} {data=<hex string>}
2	SendAlarmEvent sender=<address> text=<text> {reporter=<ID>} {notification=<ID>} {enum=<ID>} {data=<hex string>}
3	SendEvent sender=<address> text=<text> {reporter=<ID>} {notification=<ID>} {enum=<ID>} {data=<hex string>}

Where:

- The sender address is the 16-bit board ID listed as the source of the message (required)
- The text field is the free-form text to report (required)
- The reporter ID is a 32-bit ID number that identifies a type of reporter (optional, 0 by default)
- The notification ID is a 32-bit number that identifies a kind of alarm associated with the reporter (optional, 0 by default)

¹ See SV2 or SV3 Interface Control Description for message detail

- The enumerated ID is a 32-bit number that is used in combination with reporter ID and notification ID to make alarms that are set and cleared unique (optional, 0 by default)
- By convention, reporter ID values 1-999 are reserved to LRI and values 1000-1999 are reserved LRI-developed SMC software. Third-party developers should contact LRI to obtain a reserved range. The Magnetometer service is using 1010, but it's a configurable value in the XML file.

Notification ID's can be assigned by each service independently. The Magnetometer service has defined the following:

- 0 = background clock reset
- 1 = automatic clock reset
- 2 = instrument is mistuned
- 3 = received a warning from the instrument
- 4 = recieved garbage serial data
- 5 = lost serial connection to Magnetometer
- 6 = Magnetometer was hot swapped
- 7 = calibration of pressure zero was successful
- 8 = calibration of pressure zero failed

SMC services are setting the enumerated ID equal to the board address of the service, which allows alarms from difference service instances to be differentiated. For example, if there are multiple ADCP's in the same payload box, or if there are ADCP's in two different payload boxes, their alarms will not be conflated.

SWSMAC-471 "Parsing problem with RUNSHELL commands that include quotes"

Description

If a "runshell" command contains quoted items, the command line is re-assembled without quotes, which can cause errors. For example, in the case of:

```
runshell sed -i 's/val>60/val>30/g' ~/roots/200_MAM/MAMSensor-config.xml
```

The single-quotes are removed during the parsing process, resulting in the angle brackets being interpreted by the shell.

Workaround

The workaround for this is to base64 encode the data. For example, under Linux:

```
echo "sed -i 's/val>60/val>30/g' ~/roots/200_MAM/MAMSensor-config.xml" | base64 -w 200
```

(I add -w 200 to ensure that the output comes out as one line rather than multiple lines.)

The output is:

```
c2VkJC1pICdzL3ZhbD42MCM92YWw+MzAvZycgfi9yb290cy8yMDBfTUFL01BTVNlbnNvci1jb25maWcueG1sCg==
```

Use that string in the runshell command and pipe the decoded data into a shell:

```
runshell c2VkJC1pICdzL3ZhbD42MCM92YWw+MzAvZycgfi9yb290cy8yMDBfTUFL01BTVNlbnNvci1jb25maWcueG1sCg== | base64 -d | bash
```

Resolution

Used the "GetTail()" member of the parser class to return the unparsed data following token 0 as the command line.

Dependencies

This version of software requires RegulusOS 1.1.1 (P/N 05858-02)

Open Issues

SWSMAC-343 “MAM reports a non-zero peak wave direction in sensor telemetry when reporting an invalid sample (i.e. with significant wave height, peak period and average period set to zero.)”

SWSMAC-445 “MOSE did not stop sampling with MOSE --off and MOSE --stop, even though acks were returned”

SWSMAC-447 “SetupXDATA does not adequately handle encrypted partition case”

SWSMAC-460 “Implement communication channels over the expansion port interface”

SWSMAC-461 “Enable expansion port clients to receive copies of telemetry messages”

SWSMAC-462 “Enable expansion port clients to receive copies of VMC-resident low-level feeds”

SWSMAC-463 “Re-implement startup support for encrypted partitions using either expansion port or sensor port”

End-user Impacts

Write Console Script results on WGMS may look different to end users when the commands fail: they will be marked as “Nack with Description” instead of “Acked”

Command Status	Combined Acks
Nack With Description	[succeeded:false] [exception:St16invalid_argument] [e...
Acked	
Nack With Description	Caught St13runtime_error (Caught CORBA exception BadP...
Nack With Description	/bin/sh: 1: nosuchcommand: not found
Acked	Tue Jul 29 12:48:12 PDT 2014

Support Impacts

N/A

Changes in Version 2.2.0 (05/30/2014, not released)

This version (which was never released) includes some improvements for the GPSWaves module.

New Features

SWSMAC-470 “Write Console Script from SMC to other SV2 payloads”

Description

It would be useful to be able to issue a Write Console Script command to other payloads or to the Float C&C. For example, Payload Interface could implement a command like WCS <target address> <command>

Resolution

Added a new command to the SV2 configuration of PayloadInterface:

WCS <board address> <command>

The result is the ack code followed by the response. For example:

```
$ ServerExec PayloadInterface 'WCS 0 fcc -s'
ACK3D:PL1:On 0x0200(Full PIB v2.01.393) 0x2100(Sensor mgmt. compute v0.00.
001) 0x2000(Sensor mgmt. compute v2.01.000) PL2:Off PL3:Off iPIB:On 0x1000
(v1.03.350) No UnTwist
$ ServerExec PayloadInterface 'WCS 0x200 pib -c'
ACK3D:PM1=COMM PM2=SMC PM3=EMPTY Power Down Delay = 1 seconds
```

SWSMAC-473 "GPSWaves: update GPSWavesLib to use UnifiedWavesLibrary from Jim Thomson"

Description

GPSWavesLib based on UnifiedWavesLibrary to select input source(s) from system GPS and Microstrain GPS/AHRS to produce waves reports and spectrum.

Resolution

GPSWavesLib updated to use UnifiedWavesLib APRIL2014 from Jim Thomson.

Outputs now have the following additional spectrum data in the output file /mnt/XDATA/spectrum.log:

- the normalized spectral moments A1, B1, A2, B2, and spectral quality CHECK factor (ideally = 1 at all frequencies)

The following changes were made to the config file /home/smc/roots/200_GPSWaves/GPSWaves-config.xml:

- Added key <inputType> for selecting Input source for waves report and spectrum computation:
 - 0 = default (External/Microstrain GPS, NED, AHRS)
 - 1 = System GPS only
 - 2 = external (Microstrain) GPS only
 - 3 = external (Microstrain) GPS and NED
 - 4 = external (Microstrain) GPS, NED and AHRS
- Added key <AHRSorientation> with possible values:
 - 0 = Integrated mast orientation (X = down) This is the default setting
 - 1 = clamp-on housing orientation
 - 2 = orientation z = down
- Added key <AHRSpitchRoll> with possible values:
 - 0 = do not use
 - 1 = use as reported from AHRS data

Bug Fixes

SWSMAC-455 "Magnetometer interface should catch a restart or hot swap"

Description

If the Magnetometer is power cycled during operations outside of any actions taken by the software, and it's already sampling, any automated sampling mode that it's in is lost. The software should recognize when this happens and attempt to report the problem and resume sampling.

Resolution

When the Magnetometer is first powered on, it shows this:

```
'Explorer rev1.2'
```

'Serial Number 36864'
'Autotuning is ON'
'WARNING: Long deflect is forced on (cmd:k)'
'Cycling at 1.00s' xor 'Ready'

'Ready' means cycling is off. Otherwise it is cycling.

This software will catch this at "Serial Number" and run the "--start" command automatically to resume sampling.

Dependencies

This version of software requires RegulusOS 1.1.1 (P/N 05858-02)

Open Issues

SWSMAC-343 "MAM reports a non-zero peak wave direction in sensor telemetry when reporting an invalid sample (i.e. with significant wave height, peak period and average period set to zero.)"

SWSMAC-445 "MOSE did not stop sampling with MOSE --off and MOSE --stop, even though acks were returned"

SWSMAC-447 "SetupXDATA does not adequately handle encrypted partition case"

SWSMAC-460 "Implement communication channels over the expansion port interface"

SWSMAC-461 "Enable expansion port clients to receive copies of telemetry messages"

SWSMAC-462 "Enable expansion port clients to receive copies of VMC-resident low-level feeds"

SWSMAC-463 "Re-implement startup support for encrypted partitions using either expansion port or sensor port"

End-user Impacts

N/A

Support Impacts

N/A

Changes in Version 2.1.0 (04/30/2014, P/N 02617-26)

This release is targeted toward SV3 expansion port functionality that is available with Regulus release 1.0. This impacts all services a little bit because there is an optional JSON description of the commands for each service that is published to the shore-side system. The release also incorporates significant changes to the Marine Magnetics Magnetometer interface and the Mobile Controller feature, supporting Mobile WGMS.

The release requires a new OS (RegulusOS 1.1.1), which requires the payload box to be opened and the SD card to be removed and reformatted or replaced.

New Features

SWSMAC-434 "File set manager should do a "RollFiles" on process shutdown (Magnetometer)"

Description

When a Shutdown signal is received by PayloadInterface, it broadcasts a Shutdown event, which is handled by each process to close files and clean up before power the system halted and power is removed. Processes implementing a logging file set (like the Marine Magnetics Magnetometer interface) should call the RollFiles() method during the shutdown phase so that the file where new data is being logged is moved into the storage directory and optionally compressed.

Resolution

The Marine Magnetics Magnetometer interface implemented this feature.

SWSMAC-444 "Implement Gaussian filters for Magnetometer data"

Description

Implement two configurable filters for Marine Magnetics Magnetometer data to be uploaded once per minute:

1. Average of the past 60 seconds
2. Gaussian filter including 91 values (the center is approximately 46 seconds from the current time.

Resolution

The following tasks were completed:

1. Implement a moving data collection to hold up to the last 91 values output from the device.
2. Implement Gaussian filters.
3. Add settings to configuration XML (selection of filter type: 1 for average of the past 60 seconds, 2 for Gaussian filter)

```
<magDataFilterSettings class_id="17" tracking_level="0" version="1">  
  <selectedFilter>2</selectedFilter>  
</magDataFilterSettings>
```

4. Add a new field ("Filtered Mag Field (nT)") that can be added to the PayloadInterface-telemetry-config.xml to add output of the filter.

The Marine Magnetics Magnetometer interface implemented this feature.

SWSMAC-446 "Implement new telemetry output with parsed fields for Magnetometer"

Description

Include the filtered data and parse out from the Marine Magnetics Magnetometer sample into individual columns.

Resolution

A new event type was added (MagMessageEvent) that carries parsed fields. This is now used instead of StringEvent, which just reported the output from the device. The type 0x91 event now reported is sub-type 1, and the entry in PayloadInterface-telemetry-config.xml looks like this:

```
<Element>  
  <first boardID="0xFF" taskID="0x1D"/>  
  <second>  
    <key boardID="0xFF" taskID="0x1D"/>  
    <payloadType value="0x0000008C"/>
```

```

<eventNameToFormatCode>
  <Element>
    <first value="MagnetometerRawData"/>
    <second value="0"/>
  </Element>
</eventNameToFormatCode>
<reportParams>
  <Element>
    <first value="0"/>
    <second maxSamples="2" targetAddress="2">
      <reportHeaderFormat>
        <fields>
          <Element libField="Header91"/>
          <Element libField="Raw" eventField="Telem-LengthToFollow"/>
          <Element libField="Raw" eventField="Telem-PayloadType"/>
          <Element libField="Raw" eventField="Telem-BoardID"/>
          <Element libField="Raw" eventField="Telem-TaskID"/>
          <Element libField="FC0BinaryFormat1"/>
          <Element libField="RawINT8" eventField="Telem-NumSamples"/>
        </fields>
      </reportHeaderFormat>
      <sampleHeaderFormat>
        <fields>
          <Element libField="Raw" eventField="Telem-Latitude"/>
          <Element libField="Raw" eventField="Telem-Longitude"/>
          <Element libField="Raw" eventField="Telem-TimeSec"/>
          <Element libField="Raw" eventField="Filtered Mag Field (nT)"/>
          <Element libField="Raw" eventField="Date YY.JJJ"/>
          <Element libField="Raw" eventField="Time HH:MM:SS.s"/>
          <Element libField="Raw" eventField="Mag Field (nT)"/>
          <Element libField="Raw" eventField="Signal Strength"/>
          <Element libField="Raw" eventField="Depth (m)"/>
          <Element libField="Raw" eventField="Leak"/>
          <Element libField="Raw" eventField="Measurement Time (ms)"/>
          <Element libField="Raw" eventField="Signal Quality"/>
        </fields>
      </sampleHeaderFormat>
    </second>
  </Element>
</reportParams>
</second>
</Element>

```

SWSMAC-450 "Correct Magnetometer startup timing"

Description

Increase Marine Magnetics Magnetometer interface communication persistence during startup. Currently, it exits if the device does not communicate within 2 seconds.

Resolution

The startup process sends 14 '0' commands at 1-second intervals now instead of 100-millisecond intervals. The "maxTries" value was increased from 3 to 10 in the configuration XML.

SWSMAC-451 "Magnetometer: Make clock re-sync occur at midnight UTC"

Description

Currently, when the Magnetometer software makes a successful connection to the Magnetometer at start up, it sets the clock re-sync to occur at an interval specified in the configuration XML. Customer wants the clock re-sync to occur at midnight UTC. This can be done by calculating the day the specified interval lands on and then rounding up to midnight.

Resolution

1. Fixed a bug with an infinite loop introduced in a recent checkin.
2. Updated clock re-sync to occur at midnight UTC.
3. Added the detection of stale and bad magnetometer data filter states. Returns a special number "99999.999" nanoTeslas to WGMS. In the case of no data at all, e.g. probably

something is not connected or damaged, it will return the special number and empty Mag raw data fields; time and GPS will be up to date.

SWSMAC-456 "Create SV3 expansion port interface for SV3"

Description

SV3 implements a protocol over the Ethernet conductors of its expansion ports that allows clients to issue HTTP GET and multi-part POST calls to:

- Register a payload, describing its commands, software version, hardware and serial numbers
- Define a polling ID so that the payload can pick up commands waiting for it
- Reply to commands that it picks up
- Send telemetry messages back to shore
- Send files to shore via WGMS legacy file transfer protocol or the SV3 bulk transfer service
- Receive notification that the expansion port is about to be powered off and interact with the VMC to delay long enough for it shut down properly
- Invoke methods on the VMC through the same scripting interface that WGMS has access to
- Receive GPS notifications (only 2 multipurpose I/O lines are available over the expansion port and they may be dedicated to a purpose other than GPS distribution)

The SMC should be configurable to support this protocol in place of the SV2 payload protocol, which is also supported on SV3 (so the SMC must support both protocols.)

Resolution

1. Regulus implements a REST protocol with HTTP GET and POST methods to do the above:

Register payload and polling ID with optional registration group (HTTP POST, sending registration JSON): <http://192.168.103.1/xp?op=meta&id=polling-id&hostid=registration-group>

Register complete (HTTP GET):
<http://192.168.103.1/xp?op=metadone&id=polling-id>

Poll VMC (HTTP GET, reading back JSON with command to execute, GPS reading, shutdown notification or timeout):
http://192.168.103.1/xp?op=poll&id=polling-id_1&id=polling-id_n&format=nnn&gps=true&shutdown=true&maxcommands=CCC&timeout=ttt

Reply to command from VMC (HTTP POST, sending the response):
<http://192.168.103.1/xp?op=reply&id=polling-id&tag=ttt&complete=flag&success=flag>

Invoke a script command on VMC (HTTP GET, reading back the response to script command):
<http://192.168.103.1/xp?op=script&id=polling-id&cmd=command-text>

Send a file to shore using bulk transfer service (HTTP POST, sending the file name and content): <http://192.168.103.1/xp?op=upload&id=polling-id>

Send a file to shore using WGMS legacy file transfer service (HTTP POST, sending the file name and content):
<http://192.168.103.1/xp?op=uploadlegacy&id=polling-id>

Send Level 2 packet with type and packet ID (HTTP POST, sending the Level 1 content):
<http://192.168.103.1/xp?op=packet&id=polling-id&type=L2-type&wid=L2-packet-ID>

Send a raw communication packet back to shore (WGMS) (HTTP POST, sending the content): <http://192.168.103.1/xp?op=rawpacket&id=polling-id>

Set AMPS Power Switch (HTTP GET):

<http://192.168.103.1/xp?op=pwr&id=polling-id&cmd=on-or-off&sw=switch-name>

Delay Shutdown (HTTP GET):

<http://192.168.103.1/xp?op=offdelay&id=polling-id&delay=nnn&setdefault=flag>

Install JAR file (HTTP POST, sending file name and content):

<http://192.168.103.1/xp?op=jar&id=polling-id>

The Linux utility "curl" can correctly execute each scenario. For example, for HTTP GET:

```
$ 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,"seedToGoal":"NaN","sog":"NaN","fog":"NaN"}}
```

For HTTP POST:

```
$ curl -X POST -F 'file=@"meta.txt"' 'http://192.168.103.1/xp?op=meta&id=2000'
```

OK

"curl" and the interface library ("libcurl") are available on many platforms (see <http://curl.haxx.se>) For rules on formatting JSON, see <http://www.json.org/>

2. The SMC CORBA IDL module that contains payload registrations has been extended to include a new "descriptiveJSON" field.

The PayloadManagerClient class, which is instantiated by every C++ sensor interface, looks for a file in the startup directory (/home/smc/roots/nnn_servername) for a JSON file named for the CORBA name and reads the content into this field. For example, for LuaWrapper, it looks for /home/smc/roots/200_LuaWrapper/LuaWrapper.json. If that file is not found, it looks for "reg.json" in the same directory.

This file should contain, at a minimum, the list of commands to be exposed. It can also contain hardware serial numbers or other information that should be made part of the vehicle manifest. For example, here's the JSON file for PayloadInterface (/home/smc/roots/100_PayloadInterface/PayloadInterface.json):

```
{
  "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" ],
      "paramTypes" : [ "java.lang.String" ],
      "template" : "%ttrunshell %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:%ttrunshell %p",
      "timeout" : 600,
      "tooltip" : "Run shell command and return result as a file transfer"
    }
  ]
}
```

3. PayloadInterface implements a new configuration item ("sv3Enabled") which defaults to false (SV2 payload mode), but can be set to true to use the SV3 REST interface. There is also a new section devoted to settings for the SV3 interface:

```
<sv3NetInterfaceSettings class_id="3" tracking_level="0" version="2">
  <hostAddress>192.168.103.1</hostAddress>
  <hostAddressVerb>GetVMCAddress eth0</hostAddressVerb>
  <connectTimeoutSec>30</connectTimeoutSec>
  <transactionTimeoutSec>120</transactionTimeoutSec>
</sv3NetInterfaceSettings>
```

The content of the hostAddressVerb field will be used to determine the host address, if the field is not blank. "GetVMCAddress" picks up the address registered on the given adapter and changes the last octet to 1, which should usually work. Note /etc/network/interfaces must be modified to enable the network adapter during boot. On the left is the default configuration; on the right is the configuration that enables the network to start up automatically and assign an address:

```
#auto eth0                                auto eth0
iface eth0 inet dhcp                      iface eth0 inet dhcp
```

To enable automatic Ethernet, you can use the new script enable script:

```
sudo EnableAutoEthernet
```

To disable automatic Ethernet, you can use the new disable script:

```
sudo DisableAutoEthernet
```

In SV3 mode, PayloadInterface listens to the payload registration events from PayloadManager and implements a registration loop, passing registration data to the VMC using the "meta" and "metadone" operations. When all payloads are successfully registered, it implements a polling loop to listen for commands and shutdown notifications.

Note that all payloads are registered with ID's equal to the hexadecimal version of the board ID. So for example, if PayloadInterface is set up with board address 8192 (0x2000), then its polling ID is set to "2000"

If the system GPS is down, PayloadInterface also picks up GPS readings from the VMC and sends them as "HostGPSSample" (GPSListener picks this up and rebroadcasts it as "GPSSample" or "GPSSample1Sec")

When commands are picked up from the VMC, the command content is forwarded to the target server via CORBA (as it is when acting as a SV2-style payload.) Responses are returned using the "reply" operation.

The SV3 REST interface doesn't pass the function code along with the command. For SV2 payloads, function code 1 means "return the result as a file." To get the same functionality, prefix the "template" string in the command JSON with "file:" to return the result using a WGMS file transfer or "bulk:" to use the bulk transfer service.

In SV3 mode, PayloadInterface continues to listen to CORBA events and convert them to telemetry messages as directed by PayloadInterface-telemetry-config.xml, but it uses the new "rawpacket" REST operation to send the events to shore.

4. PayloadInterface implements the following new commands when configured to use the SV3 REST interface:

- **HOSTGPS** *true-or-false* forces GPS to be polled (true) or removes that override (false) Normally, PayloadInterface will only poll GPS if the system GPS failed.
- **HOSTINVOKE** [--board-address=*board-address*] *command-script* sends invokes the script (*command-script*) on the VMC and returns the literal result. If a board address is specified (must be from a registered payload), the corresponding polling ID is used; otherwise, the command appears to come from PayloadInterface.

The scripting interface from WGMS requires the invocation to start with a percent sign ("%"); when using **HOSTINVOKE**, do not specify a leading percent sign.

- **KILLPOWER** sends a command to the VMC to turn off the expansion port. The polling loop will go through a normal shutdown notification and the payload will be shut down and powered off.
- **SETAMPSSW** [--board-address=*board-address*] *switch-name* *true-or-false* powers (true) or removes power (false) from the named power switch in the payload box where the SMC is mounted. If a board address is specified (must be from a registered payload), the corresponding polling ID is used; otherwise, the command appears to come from PayloadInterface.

The switches are named ICPwr*n*, where *n* varies from 1 to the number of switches on the AMPS board (usually 6 or 8.)

These commands can be executed from the SMC command line via ServerExec; for example:

```
$ ServerExec PayloadInterface 'hostinvoke uptime.uptime()'
Uptime: 0 days, 07:26:46
```

SWSMAC-457 "Add a standard field to the GPS event that identifies its source"

Description

GPS distribution can come from multiple sources, frequently from the VMC via the expansion port or as distributed from the CCU to the SMC over expansion port aux lines or dedicated sensor port GPS/PPS lines. A new standardized field in the GPS event will indicate "host" for samples received over the expansion port (not synchronized with PPS on the SMC side) or "system" when coming over hard wires from the CCU. This field can be used in the future to indicate when the GPS source is from another payload or host-based source.

Resolution

A new named field ("GPS-SourceName") was added to the GPSSampleEvent class, which is a free-form string indicating the source of the GPS sample. GPSTListener sets it to "system" for samples from the local serial port (the spare lines on the expansion port or the dedicated GPS lines on the sensor port); PayloadInterface sets it to "host" for samples coming from the SV3 REST protocol.

Here's a sample from PayloadInterface.log showing GPS samples coming from serial port stamped as serial, then samples coming from the SV3 REST protocol when the local serial port stops sending:

```
2014/05/09 23:16:11.712230 dm3730_zoom_torpedo [1459] SV3NetChannel 0x2040 LuaWrapper
Registering payload
2014/05/09 23:16:12.746534 dm3730_zoom_torpedo [1459] SV3NetChannel 0x20F1 GPSTListener
Registering payload
2014/05/09 23:16:13.520354 dm3730_zoom_torpedo [1459] SV3NetChannel 0x20F2 PPPHost
Registering payload
2014/05/09 23:16:15.529449 dm3730_zoom_torpedo [1459] GPSSample:GPSSample/LiquidR-SMC:
[system] latitude=37.4107827 longitude=-122.0045497 numSatellites=8
2014/05/09 23:16:17.951715 dm3730_zoom_torpedo [1459] Registration loop reports all
registered
2014/05/09 23:16:17.969996 dm3730_zoom_torpedo [1459] New comm status = true
2014/05/09 23:16:17.978969 dm3730_zoom_torpedo [1459] Sending encrypted partition status
(71000000001500010000002000000186815601D603A3FBC1616D53018EBE)
2014/05/09 23:16:18.560517 dm3730_zoom_torpedo [1459] (MessageForwarder) Waiting
(communicating=true messages=0)
2014/05/09 23:16:20.488064 dm3730_zoom_torpedo [1459] GPSSample:GPSSample/LiquidR-SMC:
[system] latitude=37.4108032 longitude=-122.0045313 numSatellites=8
2014/05/09 23:16:25.480067 dm3730_zoom_torpedo [1459] GPSSample:GPSSample/LiquidR-SMC:
[system] latitude=37.4108232 longitude=-122.0044882 numSatellites=8
2014/05/09 23:16:30.479181 dm3730_zoom_torpedo [1459] GPSSample:GPSSample/LiquidR-SMC:
[system] latitude=37.4108468 longitude=-122.0044565 numSatellites=8
2014/05/09 23:18:45.467948 dm3730_zoom_torpedo [1459] GPSSample:GPSSample/LiquidR-SMC:
[system] latitude=37.4107887 longitude=-122.0044322 numSatellites=8
...
```

```
2014/05/09 23:18:58.390366 dm3730_zoom_torpedo [1459] EventToTelemetry for
SystemGPSOffline:EventBase/LiquidR-SMC
2014/05/09 23:18:58.396927 dm3730_zoom_torpedo [1459] System GPS is now offline
2014/05/09 23:19:09.986937 dm3730_zoom_torpedo [1459] GPSSample:GPSSample/LiquidR-SMC:
[host] latitude=37.410778 longitude=-122.004517 numSatellites=0
2014/05/09 23:19:14.989927 dm3730_zoom_torpedo [1459] GPSSample:GPSSample/LiquidR-SMC:
[host] latitude=37.410774 longitude=-122.004526 numSatellites=0
2014/05/09 23:19:20.017119 dm3730_zoom_torpedo [1459] GPSSample:GPSSample/LiquidR-SMC:
[host] latitude=37.410768 longitude=-122.004534 numSatellites=0
```

SWSMAC-458 "Recognize when hard-wired GPS is unavailable and automatically switch GPS over the expansion port"

Description

If there is no GPS coming in over the dedicated hard-wired port (/dev/ttyO1), GPSTListener should declare that the system GPS is down; at that point, GPS should start being polled via the expansion port and that should be used transparently until the GPSTListener finds that valid GPS samples are coming in again over the hard-wired port. At that point, GPSTListener should declare that the system GPS is up again, and GPS samples should stop being read via the expansion port.

Resolution

1. GPSTListener now notices when it isn't seeing GPS from the hard-wired port and sends a new event ("SystemGPSOffline"); this event is repeated periodically as long as no GPS is received from the port.
2. GPSTListener announces a new event ("SystemGPSOnline") when it goes from not seeing GPS to seeing it.
3. PayloadInterface initially assumes that GPS is being received by GPSTListener, but if it receives the "SystemGPSOffline" event in SV3 mode, it begins polling the GPS position using the SV3 REST interface and sends resulting GPS samples out as "HostGPSSample."
4. PayloadInterface stops polling for GPS if it receives the "SystemGPSOnline" event.
5. If the system GPS is off-line and GPSTListener receives "HostGPSSample," it re-tags the event as "GPSSample" or "GPSSample1Sec" (the standard GPS labels) and re-broadcasts it.

SWSMAC-459 "Change from structure overlays to byte streams in PayloadInterface (code robustness)"

Description

PayloadInterface was originally implemented using structure overlays to define messages, but this violates anti-aliasing rules and is fragile to changes in numeric value byte ordering and doesn't robustly check received buffer sizes. The new ByteStream class implements these checks, so PayloadInterface should be modified to use ByteStream.

Resolution

The feature was implemented and the SV2/sensor port interface was re-tested.

SWSMAC-464 "Implement ring indicator on SimpleSBDQueue"

Description

The ring indicator feature of Iridium modems outputs "SBD RING" whenever the Iridium gateway goes from having no messages pending to having at least one message pending. The modem interface program should take this as a queue to start a communication session. The feature must be provisioned on a modem-by-modem basis. Without ring indicator, the modem interface program has no way of knowing when to poll for messages from the gateway, and so usually sleeps for a configurable amount of time between poll sessions.

There is already a configuration to enable or disable the new modem interface program behavior, but currently it doesn't do anything. Implement responsiveness to the ring indicator signal when configured to use it. In summary:

1. Poll at a configurable interval, regardless of state of "use ring indicator" flag
2. If "use ring indicator" is set, and if "SBD_RING" comes in, stop waiting for the inter-session delay and start another session.

Note that to be effective, "duty cycle mode" should be false: turning off the modem will prevent it from getting the ring indication. Also, the "mailbox check interval" can be extended since usually you will not have to poll; however, if the modem doesn't receive a ring indication for some time and the reason is that the ring indication was lost, a session is needed to resume receiving ring indications.

Resolution

Implemented as above. The "useRingIndicator" flag in the configuration is now used.

SWSMAC-468 "MobileWGMS: implement a continuous comms (stay connected) feature for all radio clients"

Description

Implement a feature that lets the radio clients stay connected and provides continuous comms, without duty cycling for power, up to a maximum time limit.

Resolution

The following command was added:

MobileCtrl --connect [durationSeconds]

The duration can be up to 6 hours. This maximum value can be set in the configuration XML file as "maxStayConnectedSeconds" (21600 seconds is recommended.)

Two additional fields have been added to the configuration XML file to give more resolution to duty cycle management:

- *radioOnSeconds* is the number of seconds to leave the radio powered, including startup time (120 seconds recommended);
- *radioOffSeconds* is the number of seconds that the radio must be powered off before powering it on again (120 seconds recommended.)

The previous governing parameter, "modemCycleTimeSeconds", is deprecated.

Bug Fixes

SWSMAC-158 "makecard utility for installing the kernel and root filesystem requires full paths"

Description

The makecard utility script, which is used to install the kernel and root filesystem, requires the full path to the installation files; if a relative path is used, the script aborts when it tries to access the tar archive.

Resolution

The script has been updated to read the full path of parameters when the script is invoked.

SWSMAC-431 "Magnetometer Software receives an unexpected Null byte"

Description

Sometimes when running for long periods of time, a Null byte would appear in the middle of the periodic message from the Magnetometer. This would cause data corruption for that single 1-second update.

Note that this problem was observed when operating on a PC rather than on the target and using a variety of development hardware: it's possible that this problem will not occur as frequently on the target.

Resolution

The parsing logic was improved to check for the valid characters within the device output. If the sample is not entirely valid, it is dropped rather than reporting possibly-invalid samples.

SWSMAC-433 "Magnetometer: Recieved message of incorrect length"

Description

Include a message length check on incoming magnetic field reading messages. If the message is of an incorrect length and begins with a '*', then ignore it.

Example of a partially received message:

```
2014/01/29 12:03:30.014462 dev-VirtualBox [2306] (Mag) Received asynch message
'*14.029/20:03:33.5 F:018622.082 S:103 D:-49'
2014/01/29 12:03:30.511099 dev-VirtualBox [2306] (Mag) Received response message '38.7m
L0 0465ms Q:69 '
```

Note they were processed half a second apart; it should have been:

```
2014/01/29 12:03:30.014462 dev-VirtualBox [2306] (Mag) Received asynch message
'*14.029/20:03:33.5 F:018622.082 S:103 D:-4938.7m L0 0465ms Q:69 '
```

In this case, it might be best to ignore this data.

Resolution

A "messageLength" configuration value was added to the protocol settings: this tracks the expected length of messages. In addition, the software attempts to piece together outputs that are sequential but delayed.

SWSMAC-441 "Magnetometer Update XML version"

Description

The XML configuration file schema should be updated to allow for the new messageLength parameter.

SWSMAC-449 "Hydrocarbon software/script missing Turner C3 data and unzipped file name collisions"

Description

Two problems have been found in the Hydrocarbon software configuration:

1. The first 30 minutes of Turner C3 data in c3raw.dat are missing.
2. Files within the zip files are not named uniquely per date, so they overwrite files when unzipped rather than being tagged with appropriate dates.

SWSMAC-452 "Magnetometer: Unable to connect to Mag on startup"

Description

Sometimes at startup, the software is unable to connect to a Magnetometer. This could be worked around by restarted the SMAC until it did connect.

This is most likely a timing issue. If the Magnetometer is in a particularly noisy environment, it may report an "Instrument Mistuned" event and take 4 to 6 seconds to re-tune itself. While re-tuning, it is unable to respond.

Resolution

When the Magnetometer software started up, it would try sending the Mag some 'T' messages as part of initializing the communication. It would only try three times and the time between each T was too short.

The Magnetometer configuration can be changed to try more than three times in the <maxtimes> field.

The timeout was increased in the code to 1 second.

SWSMAC-453 “Magnetometer check for clock desync or reset and automatically resync”

Description

A Magnetometer's clock may be reset to Jan 1st, 2000. This can occur when Magnetometer's are hot swapped or perhaps in the rare case of a power outage to the device. This occurred during a wet test in Monterey Bay.

Two changes are proposed:

1. The payload software should detect when the Magnetometer's clock is out of synch and automatically resynch the clock.
2. Use the payload's timestamp in place of the Magnetometer's clock for telemetry, rounding down to the nearest minute.

Resolution

Implemented as proposed.

SWSMAC-454 “Log file set handler does not progress to the next file and stops logging if initially at exactly the maximum size”

Description

When the accumulating data file starts off or reaches exactly the maximum size (“maxFileSize” value in file set settings), the File Set manager never copies or compresses the data file. Data stopped being recording to the file set until the software could be restarted and the file was moved manually.

SWSMAC-465 “MobileWGMS: Commands contained in large messages received by radio clients are sometimes dropped”

Description

Some commands are dropped from large messages that are received from the radio clients.

SWSMAC-466 “MobileWGMS: Message header and total length incorrect from vehicle to radio clients”

Description

The VR_SHIPTOSHORE header is not being setup correctly, and the total message length is missing for messages going from the vehicle to its radio clients.

Dependencies

This version of software requires RegulusOS 1.1.1 (P/N 05858-02)

Open Issues

SWSMAC-343 “MAM reports a non-zero peak wave direction in sensor telemetry when reporting an invalid sample (i.e. with significant wave height, peak period and average period set to zero.)”

SWSMAC-445 “MOSE did not stop sampling with MOSE --off and MOSE --stop, even though acks were returned”

SWSMAC-447 “SetupXDATA does not adequately handle encrypted partition case”

SWSMAC-455 “Magnetometer interface should catch a restart or hot swap”

SWSMAC-460 “Implement communication channels over the expansion port interface”

SWSMAC-461 “Enable expansion port clients to receive copies of telemetry messages”

SWSMAC-462 “Enable expansion port clients to receive copies of VMC-resident low-level feeds”

SWSMAC-463 “Re-implement startup support for encrypted partitions using either expansion port or sensor port”

End-user Impacts

Customers upgrading from a previous version of SMC software must open the payload box, extract the micro SD card and reformat it or replace it.

Support Impacts

N/A

Changes in Version 2.0.1 (02/07/2014, P/N 02617-25)

This release is prompted by the need to ship SV2-style payloads on the SV3 platform. It also captures a new version of ADCP, the initial version of GPSWaves and some reliability fixes for Magnetometer. It is recommended that existing SV2 customers install this version to take advantage of new features and bug fixes; SV3 customers must use this version.

New Features

SWSMAC-437 “Create wave height calculator based on Microstrain 3DM-GX35 GPS (GPSWaves)”

Description

Interface Microstrain 3DM-GX35 GPS and IMU sensor and use its GPS output to compute wave height parameters (using Jim Thomson's algorithms.) Keep the software that Jim Thomson produces in a separate library so that it can be substituted separately from the main program.

Resolution

See section on GPSWaves program, below.

SWSMAC-438 "Add break control commands on LuaWrapper's serial port object"

Description

LuaWrapper provides the script with a globally accessible lua-compatible table ("SMC") with various methods, and one of these methods is NewSerialPort. The resulting lua object has a method to transmit data, but no provision was made for controlling the break signal. Implement two new methods on the serial port object:

- (1) SetBreak() – turns on the break condition for the default period of time
- (2) SetBreakTime(ms) – turns on the break condition for the given number of milliseconds ("ms")

Resolution

New methods added to serial object returned by SMC:NewSerialPort()

SWSMAC-439 "Add support for delta theta, delta velocity, orientation matrix and orientation update matrix to MicrostrainTest"

Description

It would be helpful to add the ability to read the following variables to those that can be read by MicrostrainTest:

- Delta Theta Vector
- Delta Velocity Vector
- Orientation Matrix
- Orientation Update Matrix

Resolution

The following changes were made:

1. It is now possible to specify which variables to read and the rates in a string parameter. For example:

```
AHRS:100ms,beaconedTimestamp,eulerAngles,scaledAccelerometerVector,
scaledGyroVector,scaledMagnetometerVector,orientationmatrix|GPS:4hz
,utcTime,llhPosition,nedVelocity
```

2. The following AHRS variables are supported as parameters in the spec string:

```
beaconedtimestamp
deltathetavector
deltavelocityvector
eulerangles
gpstimestamp
internaltimestamp
orientationmatrix
orientationupdatematrix
quaternion
rawaccelerometervector
rawgyrovector
rawmagnetometervector
scaledaccelerometervector
scaledgyrovector
scaledmagnetometervector
stabilizedaccelvectorup
stabilizedmagvectornorth
wrappedrawgx
```

Although it is possible to request any field, only the following are decoded:

```
beaconedtimestamp
deltathetavector
deltavelocityvector
```

eulerAngles
scaledAccelerometerVector
scaledGyroVector
scaledMagnetometerVector
orientationMatrix
orientationUpdateMatrix

The following GPS variables are supported as parameters in the spec string:

clockInfo
dopData
ecefPosition
ecefVelocity
gpsFixInfo
gpstime
hardwareStatus
llhPosition
nedVelocity
spaceVehicleInfo
utctime
wrappedDrawnMeapacket
wrappedDrawubxpacket

Although it is possible to request any field, only the following are decoded:

llhPosition
nedVelocity
utctime

The program now be invoked as follows:

(1) MicrostrainTest {CORBA-parameters} {spec-string}
(2) MicrostrainTest DECODER {--csv}
(3) MicrostrainTest ENCODER {port-name {spec-string {run-time-sec}}}
(4) MicrostrainTest TEST1 {port-name {spec-string {run-time-sec}}}

Case (1) starts the CORBA server

Case (2) begins decoding the output of the encoder (3) or the logged data from the CORBA server (1), assuming it comes in on standard input, and outputs the result on standard output

Case (3) starts listening and encodes packets as they come on, writing them to standard output

Case (4) starts listening and writes fully decoded packets to standard output

The default spec string is:

AHRS:100ms,beaconedTimestamp,eulerAngles,scaledAccelerometerVector,scaledGyroVector,scaledMagnetometerVector,orientationMatrix|GPS:4hz,utctime,llhPosition,nedVelocity

Bug Fixes

SWSMAC-401 "GPSTListener is not compatible with SV3"

Description

SV3's GPS output is different in that:

1. NMEA strings come in a different order
2. The baud rate is at 9600 rather than 4800

GPSTListener should automatically adapt to different baud rates and ordering so that the SMC can work on either SV2 or SV3 platform.

Resolution

1. The way that NMEA data is process has changed:

Before:

Serial data was received and the data was broken into lines without any assumption of the timing. It was unclear at that point which lines went together, so it was assumed that they arrived in a given order to populate the GPS event data.

Now:

Serial data is received in a packet delimited by a configurable inter-character time gap (20 milliseconds, for example.) That packet of data is assumed to be one GPS sample, including all NMEA sentences that are part of the sample. The packet is then parsed into sentences and the resulting data populates the GPS event. Now there is no need to assume an order for the sentences.

2. A new mechanism was added for trying baud rates. It assumes there is an error if there isn't a valid sentence at least every 6 seconds or whenever invalid sentences are received. The XML file now includes a configuration for baud rates to try:

```
<tryBaudRates>
  <count>3</count>
  <item_version>0</item_version>
  <item>4800</item>
  <item>9600</item>
  <item>115200</item>
</tryBaudRates>
```

3. The latest operating system (3.0 RC3) supports monitoring the GPS receive's PPS (pulse per second) output, so GPSTListener now stores the GPS reading and sets the system time on transition of the PPS line.
4. To reduce log output, a filter was added to prevent repeating error messages that have occurred within the last minute.

SWSMAC-429 "Virtual Relay does not assign the correct board ID for communication channels"

Description

Virtual Relay has a base address (for example, 0x20AA) and sequentially assigns addresses to each of its comm channels. When messages arrive tagged for 0x20AA, they are known to be management commands or responses to queries from the management processor. The first comm channel in this example should be assigned 0x20AB as its address: messages arriving for 0x20AB should be transmitted to shore via the comm channel.

However, VR is currently setting the board ID to 0 (incorrectly.) Messages arriving via the comm channel (commands) are sent to PayloadInterface tagged as 0xAB. PayloadInterface stores them waiting for the Float C&C to poll for board ID 0, which it never does.

Resolution

Resolved the issue by moving PayloadManagerClient's Register invocation before VirtualRelayProgram's Initialize invocation, so that the board ID is available to Initialize.

SWSMAC-430 "Magnetometer Software Locks Up"

Description

The Magnetometer protocol stopped executing: no more samples were being reported. In some test cases, an attempt was being made to resynchronize the Explorer's clock with the system clock and the process never completed.

Root Cause

During debugging, it looked like a synchronization object timeout was being overwritten with a very large number, and so it would essentially never return from the lock function,

where it would normally be expected to wait only 3 seconds. It turned out that a variable was incorrectly initialized (a reference was initialized with a temporary variable.)

Resolution

The variable initialization was corrected.

SWSMAC-436 “Correct use of angle brackets on include files for makepp”

Description

#include statements should use angle brackets (<>) instead of quotes (") for system include files when using makepp. Header files in double quotes are evaluated for changes and inclusions; double quoted system files sometimes cause spurious error messages to be produced.

Dependencies

This version of software requires kernel and root file system 3.0-RC3 P/N XXXXX-XX.

Open Issues

MAM reports a non-zero peak wave direction in sensor telemetry when reporting an invalid sample (i.e. with significant wave height, peak period and average period set to zero.)

End-user Impacts

Customers upgrading from a previous version of SMC software must open the payload box, extract the micro SD card and reformat it or replace it.

Support Impacts

N/A

Installation/Upgrade Instructions

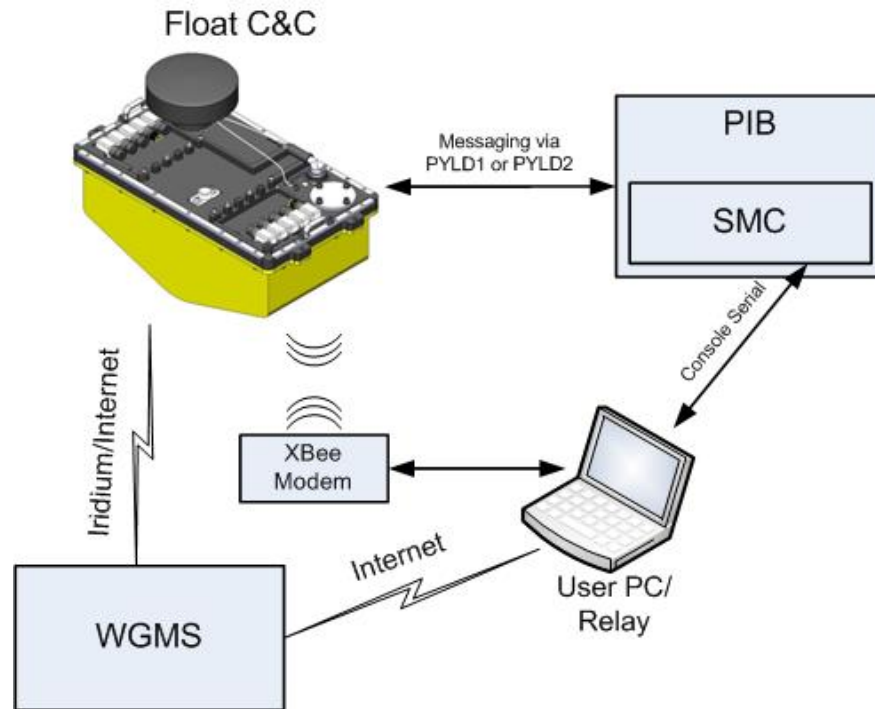


Figure 1 SV2 SMC interconnection with Float C&C and user PC (setup configuration)

WGMS (Wave Glider Management System) is a web-based application and database that provides a command and telemetry interface to vehicles. It communicates to vehicles via the Iridium network, but extends its reach via software relays.

The Float C&C is the “command and control” computer that supervises navigation and acts as a message router for commands and data for the vessel. It also implements the Iridium communication channel for access at sea, and allows for low-cost short range communication via XBee for development, setup and troubleshooting. The XBee relay (P/N 00775-XX) also exposes a terminal interface for debug message display and entering low-level commands.

The SMC implements additional communication channels, but for this setup case, communication will be via the Float C&C and XBee or via the SMC’s USB-based console.

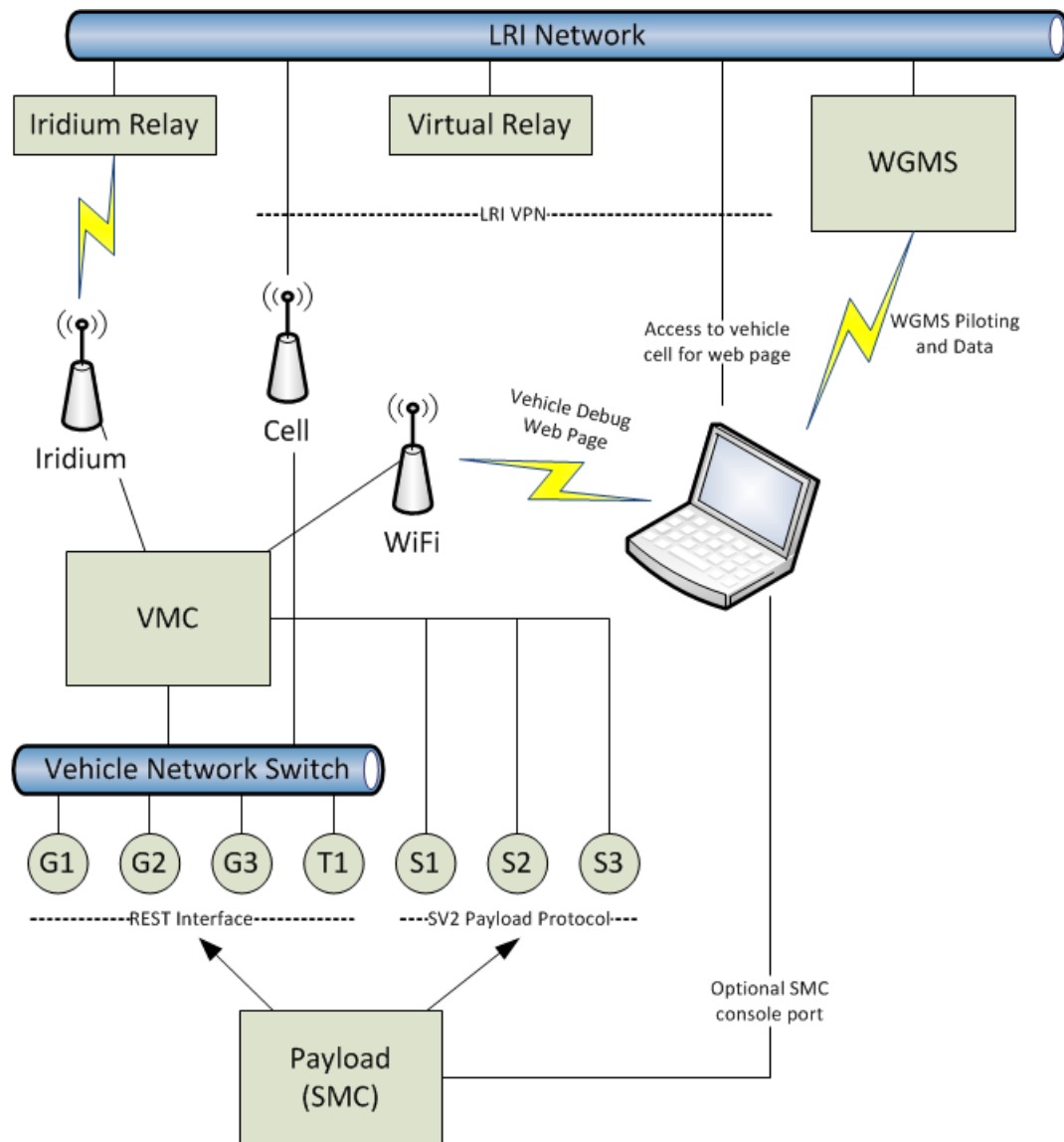


Figure 2 SV3 SMC interconnection with Float C&C and user PC

In the SV3 configuration, the SMC can interface with the VMC (Vehicle Management Computer, the equivalent of the SV2 Float C&C) in the same way that it does on SV2 using Payload Protocol over a Sensor Port; but SV3 Sensor Ports don't include umbilical aux lines, so it's not possible to talk to devices attached to the Glider or in a towfish.

It can also be connected via an Expansion Port, which provides Ethernet to the payload. The VMC exposes a REST interface with same kind of operations available through Payload Protocol. Expansion Ports carry programmable spare lines that can be used to carry GPS/PPS or umbilical aux lines.

Expansion ports also carry AMPS communication lines. AMPS is dedicated to controlling and reporting on the power components: generators, batteries, consumers, power supplies and switches. SV3 payload boxes may have power switches that switch unregulated V_{consumer} power or regulators selected for the

particular application. Currently, AMPS components can only be interfaced from the VMC or via its interfaces.

Note that SV3 does not implement an XBee radio. Instead, it implements a web page with diagnostics and setup capabilities accessible via WiFi or the cell modem. Use of the cell modem requires VPN credentials.

Setting up the SD Card

The SD card part (02574-XX) is created by setting up 3 partitions: BOOT, LINUX and XDATA:

- The BOOT partition contains instructions to the Logic Loader software (which is programmed on the CPU) as well as U-Boot and the Linux kernel.
- The LINUX partition contains the Linux root filesystem and any unencrypted SMC application files.
- The XDATA partition contains log files and resulting SMC data. XDATA can be configured as an encrypted filesystem, in which case the majority of the application is also loaded on XDATA.

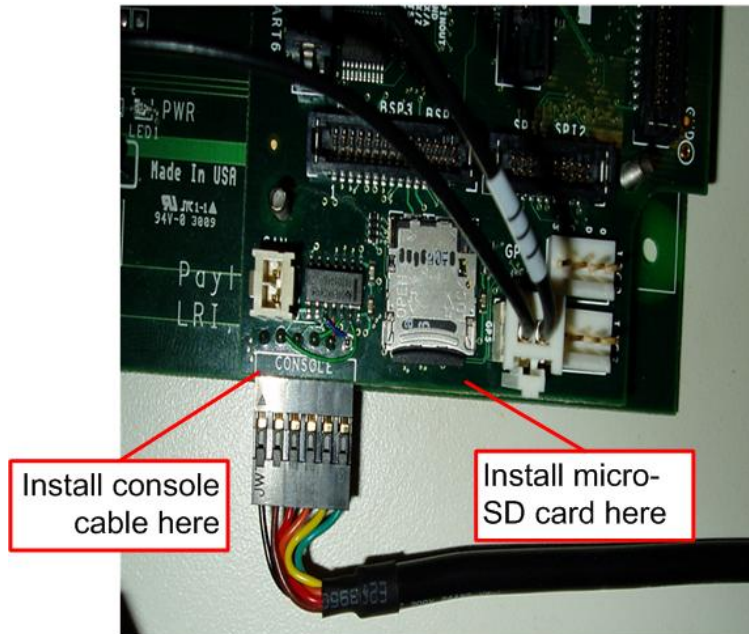
The decision of whether to encrypt the XDATA partition is typically made when creating the SD card. For systems requiring the security of an encrypted partition, care should be taken not to store anything sensitive – even temporarily – on the unencrypted (LINUX) partition.

Working with the SMC and the SV2 Float C&C

The PIB (Payload Interface Board) is a general purpose controller used to implement protocols to sensors or other computers. It has three (3) interface ports with their own power switches and serial ports. Special purpose interface boards (PCOMM's) are installed depending on the hardware protocol for each port: RS-232, RS-422 or RS-485.

The SMC can be mounted in place of one of the interface boards (two ports are consumed because of the size of the SMC.) To mount a SMC:

1. Ensure that the Float C&C payload port is powered off. This can be accomplished via the XBee Relay's debug console using command #29 (PYLD1) or #31 (PYLD2) or by issuing commands via WGMS.
2. Disconnect the PIB from the payload port (PYLD1 or PYLD2)
3. Install the SMC on PIB port J2; the PIB must be loaded with software version 2.0.335 (P/N 02639-01) or greater.
4. Reconnect the PIB to the appropriate payload port.



5. With an open payload box or a standalone SMC board, interconnect the SMC with a PC using the USB console cable (P/N 02638). When using a payload box that includes an external debug port, use an external debug cable (P/N 02824).
6. Ensure that a micro-SD card with the correct kernel version has been installed in the SMC.
7. Open the USB serial port provided by the console cable at 115,200 baud using a serial terminal like HyperTerminal or putty (Windows) or GkTerm, minicom or screen (Linux).
8. From WGMS, use the Set Parameter command to change the Payload Power On Enumeration Delay appropriately (120 seconds recommended.) This enables the SMC software to start up in time to respond to the Float C&C's enumeration request.
9. Enable power to the payload from the XBee Relay's debug console using command #28 (PYLD1) or #30 (PYLD2) or by issuing commands from WGMS. Momentarily, the LED's on the board turn on and messages from the Linux startup process begin to appear from the SMC console port. Finally, a Linux login prompt appears.
10. Watch the XBee Console for the response to the enumeration request. It should look something like this:

```
Description: Full PIB
Address: 0100 Major: 2 Minor: 1 Build: 368
Payload 1 Port

Description: SMC/Encrypted
Address: 2000 Major: 1 Minor: 1 Build: 5
Payload 1 Port
```

If you don't see a response from the SMC, one of the following may be true:

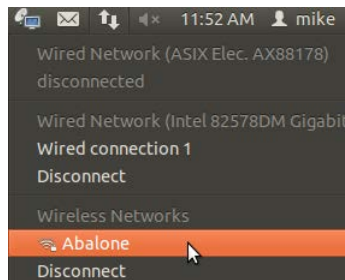
- There is no SMC software loaded.

- The PIB has not been configured to talk to the SMC at that particular address. See the user manual for the PIB for instructions on how to configure it.
- The wrong type or version of PIB software has been installed. In this case, copy the enumeration response and send it to LRI customer support for advice.
- The enumeration timeout is too short. Issuing the power on command again without powering it off first will restart the enumeration process; if the SMC then responds, try increasing the power on enumeration delay.

Note that it is important for the SMC to respond since the Float C&C uses enumeration responses to determine which addresses to poll. If the SMC's address is not polled, it cannot send any messages to the Float C&C.

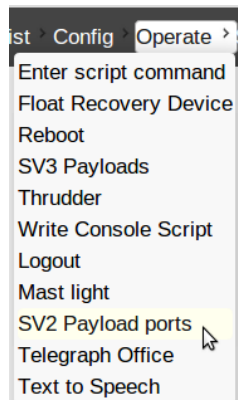
Working with the SMC and the SV3 via Sensor Port

SV2 payloads work similarly on SV3 to how they worked on SV2, but the diagnostics and operations can be performed via the vehicle's web page. The vehicle acts as a WiFi server with a SSID similar to the vehicle name. My test vehicle is called Abalone.

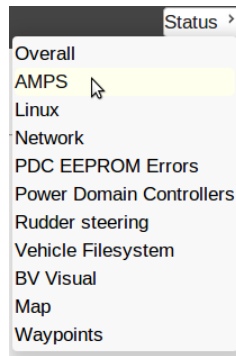


WiFi clients are assigned an IP address in the 192.168.100.X domain; the vehicle can be reached at 192.168.100.1. You should have received login credentials with your vehicle; otherwise, call Customer Support for a web page login.

The opening page shows the status of various services and devices. Pull down the "Operate" menu and select "SV2 Payload ports"1:



To open the AMPS page, pull down the “Status” menu and select “AMPS”:



Scroll down to the “Configure Expansion Switches” link (typically under slot #34) and click the link:

- **Slot 34:** Expansion Switch Module
 Firmware Version: 0x000035f1, Boot Loader Version: 0x0000381e
 Mfg. S/N=abalone434
 HW ID=0x002c001b3533363305473131
 Reported Arena Part Number = 4572A (matches manifest)
 Module Type = 7
 Found in Manifest by Serial Number.
[Configure Expansion Switches](#)
 - #1: Umbilical Expansion Switch: Not Connected
 - #2: Debug Expansion Switch: Not Connected
 - #3: G1 Expansion Switch: Connected to GPS/PPS Signals
 - #4: G2 Expansion Switch: Connected to GPS/PPS Signals
 - #5: G3 Expansion Switch: Connected to GPS/PPS Signals

The switch configuration page is displayed; make any desired changes in mapping and then click the “Configure Expansion Switches” button.

Expansion Switch Configuration

When configuring the Aux Lines for the expansion ports, the options available will depend on the port in question. At this point, Any (and ALL) expansion ports may use the GPS/PPS option regardless of the configuration of other ports. The AUX lines from only one expansion port may be connected to the Umbilical at any time (If G1 is using that option; then G2, and G3 CAN NOT be configured use it) If any of these conditions are not true, then the settings will not be updated.

Umbilical Aux Line Settings

☒ Aux Lines Are Not Used

☐ Someone (G1, G2, G3, or Debug) is Connecting to Me

☐ Aux Lines Connect To GPS and PPS signals

☐ Aux Lines Connect To the Serial Port on the VMC

G1 Aux Line Settings

☐ Aux Lines Are Not Used

☒ Aux Lines Connect To GPS and PPS signals

☐ Aux Lines Connect To The Umbilical

G2 Aux Line Settings

☐ Aux Lines Are Not Used

☒ Aux Lines Connect To GPS and PPS signals

☐ Aux Lines Connect To The Umbilical

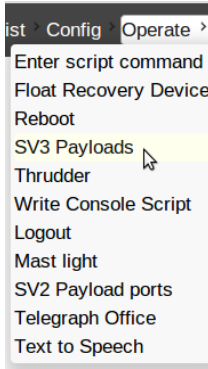
G3 Aux Line Settings

☐ Aux Lines Are Not Used

☒ Aux Lines Connect To GPS and PPS signals

☐ Aux Lines Connect To The Umbilical/Towbody

Once the setting is correct, pull down the “Operate” menu and select “SV3 Payloads”



The SV3 Payloads page has a section for each expansion port:

- When using a payload box that contains an AMPS domain, click the “Turn On SV3 Integrator Payload” button. This not only powers on the payload box, but also looks for the AMPS domain and determines if specific power switches should be turned on. For example, is an SMC known to be configured on power switch #2? If so, the appropriate switches are set.
- If no AMPS domain is in the box, click the “Turn On Port” button to apply power.
- When turning off the payload, click “Unplug Integrator Payload” to not only turn off power but also forget the registration data (in preparation for disconnecting the payload box from the VMC.)
- To simply turn the power off, click “Turn Off Port.” Note that either power off button gives smart payloads the opportunity to shut down gracefully.

Refresh			
Expansion Ports			
T1 is On	Turn Off Port	Unplug Integrator Payload	
G2 is On	Turn Off Port	Unplug Integrator Payload	
G3 is Off	Turn On Port	Turn On SV3 Integrator Payload	Unplug Integrator Payload
G0 is Off	Turn On Port	Turn On SV3 Integrator Payload	Unplug Integrator Payload
G1 is Off	Turn On Port	Turn On SV3 Integrator Payload	Unplug Integrator Payload

If one or more payloads are registered using the expansion port REST interface, the detail is displayed graphically.

G2_PayloadInterface 192_168_105_103_2000	Controlled by G2 On		
	ui template %tgetmodules method GetModules label Get Modules timeout 600 tooltip Report the configured modules running on the SMC	template %strunshell %p paramLabels [Command] method RunShell label Run Shell Command paramTypes [java.lang.String] timeout 600 tooltip Run shell command and return result as a console ack	template file:%strunshell %p paramLabels [Command] method RunShellReturnFile label Run Shell (Return File) paramTypes [java.lang.String] timeout 600 tooltip Run shell command and return result as a file transfer
reportingAddress 892 description Payload interface tag G2_PayloadInterface localTag PayloadInterface boardAddress 0 corbaName PayloadInterface hostID 1023742 SOMDM3730-20-2780AGCR-C 2013A02943 interface [swver 1.4.0]			
Configuration Parameters			
G2_GPSListener 192_168_105_103_2001	Controlled by G2 On		
	ui template %tgetmodules method GetModules label Get Modules timeout 600 tooltip Report the configured modules running on the SMC	template %strunshell %p paramLabels [Command] method RunShell label Run Shell Command paramTypes [java.lang.String] timeout 600 tooltip Run shell command and return result as a console ack	template file:%strunshell %p paramLabels [Command] method RunShellReturnFile label Run Shell (Return File) paramTypes [java.lang.String] timeout 600 tooltip Run shell command and return result as a file transfer
reportingAddress 892 localTag PayloadInterface tag G2_PayloadInterface description Payload interface boardAddress 0 corbaName PayloadInterface hostID 1023742 SOMDM3730-20-2780AGCR-C 2013A02943 interface [swver 1.4.0]			
G2_GPSListener 192_168_105_103_2001	Controlled by G2 On		
	ui template %tsgs autostart method GetAutoStart label Get Auto Start timeout 30 tooltip Determine if GPS logging starts automatically or requires a manual command	template %tsgs defaultoutput method GetDefaultOutputFile label Get Default Output File timeout 30 tooltip Get the GPS log file name used by default	template %tsgs output method GetOutputFile label Get Output File timeout 30 tooltip Get the GPS log file name currently in use
	template %tsgs autostart %p paramLabels [New State] method SetAutoStart label Set Auto Start paramTypes [java.lang.String] timeout 30 tooltip Set whether GPS logging starts automatically or requires manual command	template %tsgs defaultoutput %p paramLabels [File Name] method SetDefaultOutputFile label Set Default Output File paramTypes [java.lang.String] timeout 30 tooltip Set the GPS log file name used by default	template %tsgs output %p paramLabels [File Name] method SetOutputFile label Set Output File paramTypes [java.lang.String] timeout 30 tooltip Set the GPS log file name to be used
reportingAddress 8433			
template %tsgs start method StartGPSLogging label Start GPS Logging timeout 30 tooltip Start GPS logging to a file template %tsgs stop method StopGPSLogging label Stop GPS Logging timeout 30 tooltip Stop GPS logging to a file			

Installing the Application

- Before starting, put any software archives to be installed in a convenient place to be accessed. With the SD card inserted into a card reader and connected to Ubuntu:
 - For systems without an encrypted filesystem, copy the archives to a convenient location like /root

```
sudo cp tarball-filename /media/LINUX/root
```

```
sudo mount /media/*
```
 - For systems using an encrypted filesystem, the archives should never be stored on the SD card unencrypted. In this case, mount the encrypted filesystem and copy the archive to the mount point

```
echo 'passphrase' | sudo cryptsetup create --cipher aes-cbc-essiv:sha256 --key-size 256 --hash sha256 secsd /dev/sdd3
```

```
sudo mount /dev/mapper/secsd /mnt/secsd
```

```
sudo cp tarball-filename /media/mapper/secsd
```

```
sudo umount /mnt/secsd
```

```
sudo cryptsetup remove secsd
```

```
sudo umount /media/*
```

Note that the micro SD card device (/dev/sdd in my example) will vary depending on what other devices you have installed.

Be sure to wait for the umount command to complete before removing the micro SD card.

2. Insert the micro SD card into the SMC and power it up.
3. Log in as root
4. Set the password for the root account using **passwd**
5. Create the **smc** account (**useradd smc**) and enable access to serial ports (**usermod -a -G dialout smc**)
6. Set the password for the **smc** account (**passwd smc**)
7. Create the **smc0** account (**useradd smc0**), enable access to serial ports (**usermod -a -G dialout smc0**) and set the password (**passwd smc0**)
8. For systems using encryption, mount the XDATA partition under /home/smc:


```
echo 'passphrase' | cryptsetup create --cipher aes-cbc-essiv:sha256 --key-size 256 --hash sha256 secsd /dev/mmcblk0p3

mount /dev/mapper/secsd /home/smc
```
9. Install the application archive(s):


```
tar xvpf /path-to-archive/tarball-filename -C /
```
10. Exit the **root** user and log in as **smc**
11. For systems that do not use filesystem encryption, mount the XDATA partition and set it up:


```
sudo mount /dev/mmcblk0p3 /mnt/XDATA
sudo bin/SetupXDATA
```
12. The startup script (/home/smc/startup) configures the transceivers for some of the serial ports for different hardware protocols. Change the line that with "InitOctalUartLines" to match the desired serial port configuration of ports /dev/ttyLX2-/dev/ttyLX7. For each port, enter 'rs232', 'rs422' or 'off'; there should be a total of six settings. For example, the following enable rs232 on /dev/ttyLX2 and /dev/ttyLX3, but /dev/ttyLX4-7 are not disabled:


```
sudo InitOctalUartLines rs232 rs232 off off off off
```
13. The /home/smc/roots directory contains a sub-directory for each application instance to be executed. By default, it contains one entry for each application, so the content must be customized for each shipping configuration.

```
smc@omap35x_torpedo:~/roots$ ls -l
total 76
drwxr-xr-x 2 smc smc 4096 Jan 1 2000 001_omniNames
drwxr-xr-x 2 smc smc 4096 Jan 1 2000 002_EventManager
drwxr-xr-x 2 smc smc 4096 Jan 1 2000 050_PayloadManager
drwxr-xr-x 2 smc smc 4096 Jan 1 2000 100_PayloadInterface
drwxr-xr-x 2 smc smc 4096 Aug 23 01:09 190_BenthosModem
drwxr-xr-x 2 smc smc 4096 Jan 1 2000 200_GPSListener
drwxr-xr-x 2 smc smc 4096 Jan 1 2000 200_MAM
drwxr-xr-x 4 smc smc 4096 Aug 23 03:18 200_PPPHost
-rw-r--r-- 1 smc smc 1746 Jan 1 2000 AddressSummary.txt
```

Sample listing of configuration directories under /home/smc/roots

The applications are run in sorted order, and there is a three-digit prefix that is used by convention to establish run groupings:

- 001 is reserved for omniORB, which must run first so that applications can use CORBA services.
- 002 is reserved for EventManager, which must be available for applications that send or receive events.
- 050 is reserved for PayloadManager, which must be available for applications that register an enumerable name; since PayloadManager fires events to indicate when new names are registered, it must be after EventManager.

Other similar services can be registered in the 50-99 range.

- 100 is used by PayloadInterface, which implements the communication protocol with the Float C&C. Since some applications make use of PayloadInterface services, it makes sense for it to be available before they start.

Other similar services can be registered in the 100-199 range.

- 200 is used by most other applications. Some of these probably will not be used; services like GPSTListener (which creates events from the GPS receiver) or PPPHost (which offers communication via RUDICS) will likely be required, depending on the hardware installed on the system.

By default, component directories contain a ".disabled" file that prevents them from starting. Delete this file to enable applications to start up. If you are installing a preconfigured archive, this will have already been done.

14. Additional component configuration steps are required for some applications (see below) The **startup** script loads the octal UART driver; it may be necessary to execute it and then run **sudo killsmc** to
15. Restart the system to run the default scenario (`sudo shutdown -r now`)

Changing Encryption Passphrases

Filesystem encryption works by performing all reads and writes using the passphrase provided: the data on the storage medium is actually changed. It's not like a password that lets you in the door: it's more like a filter. Therefore, changing the passphrase is easy. It means reading the entire partition using the old passphrase to decrypt it, reformatting the partition using the new passphrase and then rewriting the partition data. For example:

1. Insert the micro SD card into to a reader connected to Ubuntu
2. Mount the old partition

```
echo 'old-passphrase' | sudo cryptsetup create --cipher
aes-cbc-essiv:sha256 --key-size 256 --hash sha256 secsd
/dev/sdd3

sudo mount /dev/mapper/secsd /mnt/secsd
```
3. Copy the partition data to a location elsewhere

```
pushd /mnt/secsd

sudo tar cvf /path-to-other-location/save-archive .

popd
```

4. Unmount the partition

```
sudo umount /mnt/secsd  
sudo cryptsetup remove secsd
```

5. Reformat the partition

```
echo 'new-passphrase' | sudo cryptsetup create --cipher  
aes-cbc-essiv:sha256 --key-size 256 --hash sha256 secsd  
/dev/sdd3  
  
sudo mkfs.ext3 -L XDATA /dev/mapper/secsd  
sudo tune2fs -c 0 /dev/mapper/secsd
```

6. Re-mount the partition

```
sudo mount /dev/mapper/secsd /mnt/secsd
```

7. Copy the partition data back:

```
pushd /mnt/secsd  
sudo tar xvf /path-to-other-location/save-archive  
popd
```

8. Unmount the partition again

```
sudo umount /mnt/secsd  
sudo cryptsetup remove secsd
```

Component Configuration

Each component or application instance to be started automatically has a corresponding subdirectory under `/home/smc/roots`. These directories include:

- A **startup** script that manages the startup process for the component.
- One or more XML-based configuration files specific to the executable.
- Other scripts used by the application.
- Data sub-directories or links to directories where data is stored.

The **startup** script is typically responsible for:

- Performing any initial cleanup from previous application runs.
- Executing the application, including any parameters to be passed.
- Setting up logging for applications that produce logs on standard output.
- Adding entries to the smc user's crontab to allow for cron-based callbacks.
- Performing any initial commands to configure behavior.

The XML configuration file (typically named *application-config.xml*) contains various settings like:

- Serial port name and parameters for serial-based sensors.
- CORBA parameters, like the name that the application will register and the names of other applications with which it will interact.
- Command verbs (what shell command to execute in a particular circumstance, like powering a sensor on or off)
- Event names for events that will be produced and ones that the application is looking for.
- Timing and retry limits

Most of these parameters will work without modification, but some exceptions are:

- Serial port names and references to electrical switches (these should be checked vs. the wiring of your system)
- The board address associated with each process, if the SMC is not configured to report as address 2000 (hex)
- References to external systems and directories that may vary by Customer or which must be unique per vehicle
- Different use cases that affect timing values
- Event naming: in one case, component A produces "Event1" and component B processes it; in another case, component A produces "Event1" that component C processes and merges with other data and produces "Event2", which component B consumes.
- Configuration of services used in common by multiple applications.

Each component can also have a JSON registration file that is rolled up into the manifest file. The JSON file should be named according to the CORBA server name; for example, the Payload Interface process has a CORBA server name of "PayloadInterface," so its JSON description file is "PayloadInterface.json"

The content of JSON file, at a minimum, the list of commands that are exposed by the process, argument names and types, tool tip descriptions and a string template that describes how the arguments will be formatted if the command is invoked. For example, the following is the JSON published by Payload Interface:

```
{
  "ui" : [
    {
      "label" : "Get Modules",
      "method" : "GetModules",
      "template" : "%tgetmodules",
      "timeout" : 600,
      "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"
    }
  ]
}
```

In the above:

- *label* is a text string to put on a button on the UI;
- *method* is the name sent back to the VMC when the command is invoked (the method must not include spaces);
- *paramLabels* is an array of strings describing each parameter to the command;

- *paramTypes* is an array of strings that describe the type of each command parameter, 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 (<i>nn</i>) is specified, it indicates which parameter to expand; otherwise, all parameters are expanded. Other format character (<i>ff</i>) are as follows: <table><tr><td>*</td><td>Use spaces to separate parameters (default)</td></tr><tr><td>,</td><td>Use commas to separate parameters</td></tr><tr><td>C</td><td>Quote parameters using C language standards (defaults to shell rules)</td></tr><tr><td>U</td><td>Quote parameters using URL rules (defaults to shell rules)</td></tr></table>	*	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)
*	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 to give more information about the command when the mouse hovers over the button (or whenever it would be appropriate.)

See the SV3 ICD for additional detail.

Aquatracka (Chelsea Technologies Group) Configuration

Aquatracka implements a CORBA command and event interface to a Chelsea Aquatracka fluorimeter. This section indicates what has changed and gives some setup guidance, but please consult the LRI Aquatracka Sensor User’s Guide for additional detail.

Commands

Commands to Aquatracka are sent to process name “Aquatracka” with ServerExec or it is typically assigned an address of 0x2016 (the upper byte varies with the base address of the SMC.) The available commands are:

AQUAT --start

Wakes up the sensor and begins streaming data.

AQUAT --stop

Stops the sensor from streaming data (sensor is still powered on)

AQUAT --run *command-file*

Runs a set of commands in the given command file.

AQUAT [-o | --output] *output-file*

Directs raw output from the sensor to the given output file.

AQUAT --off

Switches off power to the sensor.

AQUAT --on

Switches on power to the sensor.

AQUAT --getconfig

Reads the configuration from the sensor.

AQUAT --header

Reads sensor header information (serial number and configuration constants.)

AQUAT --pause

In data collection mode, halts reporting of sensor data pending a resume command (power draw remains at data acquisition level.)

AQUAT --resume

In data collection paused mode, resumes reporting of sensor data.

AQUAT --verbose

Places the sensor in verbose reporting mode.

AQUAT --noverbose

Places the sensor in non-verbose reporting mode.

AQUAT --temperature

Polls for the temperature, which is reported in degrees (C) The sensor must have already been started.

AQUAT --concentration

Polls the Fluorophor concentration (gm/cc); the sensor must have already been started.

AQUAT --calibtemp

Turns off power to the sensor by executing the power off verb from the XML file.

AQUAT --calibref *n1 n2 n3*

Writes calibration reference values into Calibration-config.xml

AQUAT --calibsig *n1 n2 n3*

Writes signal calibration constants into Calibration-config.xml

AQUAT –calibserial *newvalue*

Writes sensor serial number into Calibration-config.xml

AQUAT –calibtemp *n1 n2 n3*

Writes temperature calibration constants into Calibration-config.xml

SELECT [-t | --time | -s | --samples | -d | --distance] *N*

Configures the way that samples are selected for export (see EXPORT command)

EXPORT [–count] [-o | --output *output-filename*]

Configures the name of the file where selected samples are written.

BenthosModem Configuration

BenthosModem implements a command interface and a pass-through data pseudo-port for use with Benthos acoustic modems. There are at least three use cases covered:

1. Acoustic range detection and communication test: this is initiated by the 'BAM START-RANGING *target-address ping-interval*' command (terminated by 'BAM STOP-RANGING') and results are reported via telemetry. This operation may be needed to locate the sea floor modem and calibrate communication parameters.
2. Pass-through mode: a client application sends and receives data via the local modem and the acoustic link to device connected to a remote (sea floor) modem. Data to be sent to the remote device are written to a pseudo port; data received from the remote device can be read from a pseudo port.
3. Collection of recorded data: a client program or script issues commands like 'BAM READDATARECORDER *target-address block-range*' to retrieve data stored in a remote (sea floor) modem's data recorder; this data has been passively-received and stored by a sea floor modem operating in low power mode.

Configuring the Client Program

For clients that operate in pass-through mode, verify that the pseudo-port name (typically /home/smc/roots/190_BenthosModem/pseudotty) is used in place of a serial port. In addition, the client program will need to invoke commands (for example, 'BAM CONNECT *target-address*') on the BenthosModem process in order to establish connection.

Loading the Modem Firmware

The BenthosModem process was written and tested with modem firmware version 8.2.2. Note that sea floor modems using the older generation of firmware (6.x.x) are incompatible with later generations (8.x.x), so the set should be upgraded together.

To read the current version of installed firmware:

1. Connect the modem serial port to an available PC serial port or verify connection to the SMC

2. Open the USB serial port provided by the console cable at 9600 baud using a serial terminal like HyperTerminal or putty (Windows) or GkTerm, minicom or screen (Linux)
3. Disable hardware handshaking on the terminal program
4. Enable power to the modem; for example, on the SMC: `sudo SetPowerSwitch 2 1` (modify the switch number as necessary)
5. Wait for the chirps from the modem to indicate that it has powered up; a will be displayed indicating the current firmware version:

```
Teledyne Benthos ATM-900 Series
LF Frequency Band
Standard version 8.2.2
Aug 18 2011 19:26:49
```

Configuration BOM P/N 02690 lists the modem firmware upgrade kits that are available to use, or contact Teledyne Benthos for instructions and additional firmware.

Configuring Modem Parameters

Once the modem's firmware is set, the operational parameters should be set as follows:

1. Connect the modem serial port to an available PC serial port or verify connection to the SMC
2. Open the USB serial port provided by the console cable at 9600 baud using a serial terminal like HyperTerminal or putty (Windows) or GkTerm, minicom or screen (Linux)
3. Disable hardware handshaking on the terminal program
4. Enable power to the modem; for example, on the SMC: `sudo SetPowerSwitch 2 1` (modify the switch number as necessary)
5. After the startup banner is displayed, the modem either responds with a command prompt or a 'CONNECT' message. 'CONNECT' indicates that modem is in pass-through mode: any characters typed will be transmitted. To get back into command mode, type +++ (three plus signs) quickly.
6. At the command prompt, issue the following commands for a top side modem:

```
>AT&F
Factory Reset
user:5>@prompt=1
Prompt | 1
>ATS8=2
OK
>ATS10=255
OK
>ATS13=3
OK
>ATS15=0
OK
>@strictat=ena
Entering strictat mode
```

```

StrictAT      | Ena
>AT&W
Sregisters Stored
>

```

For sea floor modems, use the following script:

```

>AT&F
Factory Reset
user:5>@prompt=1
Prompt      | 1
>ATS13=0
OK
>ATS15=1
OK                                     (startup mode: 1=online, 2=data logger)
>ATS18=nnn
OK                                     (nnn = the address of the sea floor modem)
>@strictat=ena
Entering strictat mode
StrictAT      | Ena
>AT&W
Sregisters Stored
>

```

C3 (Turner Designs) Configuration

C3 is a command and event interface to a Turner C3 fluorometer; this is a re-implementation of the PIB-based C3 interface, and the commands are kept similar for continuity.

Commands

Commands to C3 are sent to process name "C3" with ServerExec or it is typically assigned an address of 0x2023 (the upper byte varies with the base address of the SMC.) The available commands are:

```

C3 -o
      Turns on the power switch associated with the sensor

C3 -f
      Turns off the power switch associated with the sensor

C3 -r
      Alias for C3 -o (turns on the power switch associated with the sensor)

C3 -x
      Alias for C3 -f (turns off the power switch associated with the sensor)

```

Events and Telemetry

The C3Event type contains the following name-value pair items:

```

"C3EventName" = C3Event_C3SampleHeader or C3Event_C3SampleData
"Header" = header string received from sensor
"Date" = date value (format mm/dd/yyyy) in the sample data received from sensor
"Time" = time value (format hh:mm:ss) in the sample data received from sensor
"DataC1" = first data item in the sample data from sensor
"DataC2" = second data item in the sample data from sensor

```

"DataC3" = third data item in the sample data from sensor
 "Pressure" = fourth data item in the sample data from sensor
 "Temperature" = fifth data item in the sample data from sensor
 "CompressedData" = all five data items packed into an array of 14 bytes as follows:
 first, second, third, and fourth data items represented as 3 bytes each where the first
 two bytes are the mantissa value and the third byte is the decimal value * 100, followed
 by the fifth data item * 100 represented as a two byte value.

The PayloadInterface-telemetry-config.xml entry for Turner C3 is:

```
<Element>
  <first boardID="0x20" taskID="0x17" />
  <second>
    <key boardID="0x20" taskID="0x17" />
    <payloadType value="0x00000060" />
    <eventNameToFormatCode>
      <Element>
        <first value="C3SampleHeader" />
        <second value="0" />
      </Element>
      <Element>
        <first value="C3SampleData" />
        <second value="1" />
      </Element>
    </eventNameToFormatCode>
    <reportParams>
      <Element>
        <first value="0" />
        <second maxSamples="1" targetAddress="2">
          <reportHeaderFormat>
            <fields>
              <Element libField="Header91" />
              <Element libField="Raw" eventField="Telem-LengthToFollow" />
              <Element libField="Raw" eventField="Telem-PayloadType" />
              <Element libField="Raw" eventField="Telem-BoardID" />
              <Element libField="Raw" eventField="Telem-TaskID" />
              <Element libField="FC0ASCIINormal" />
              <Element libField="RawINT8" eventField="Telem-NumSamples" />
            </fields>
          </reportHeaderFormat>
          <sampleHeaderFormat>
            <fields>
              <Element libField="Raw" eventField="Telem-Latitude" />
              <Element libField="Raw" eventField="Telem-Longitude" />
              <Element libField="Raw" eventField="Telem-TimeSec" />
              <Element libField="Raw" eventField="C3Event_Header" />
            </fields>
          </sampleHeaderFormat>
        </second>
      </Element>
    </Element>
    <first value="1" />
    <second maxSamples="7" targetAddress="2">
      <reportHeaderFormat>
        <fields>
          <Element libField="Header91" />
          <Element libField="Raw" eventField="Telem-LengthToFollow" />
          <Element libField="Raw" eventField="Telem-PayloadType" />
          <Element libField="Raw" eventField="Telem-BoardID" />
          <Element libField="Raw" eventField="Telem-TaskID" />
          <Element libField="FC0BinaryFormat9" />
          <Element libField="RawINT8" eventField="Telem-NumSamples" />
        </fields>
      </reportHeaderFormat>
      <sampleHeaderFormat>
        <fields>
          <Element libField="Raw" eventField="Telem-Latitude" />
          <Element libField="Raw" eventField="Telem-Longitude" />
          <Element libField="Raw" eventField="Telem-TimeSec" />
          <Element libField="Raw" eventField="C3Event_CompressedData" />
        </fields>
      </sampleHeaderFormat>
    </second>
  </second>
</Element>
```

```
        </sampleHeaderFormat>
    </second>
</Element>
</reportParams>
</second>
</Element>
```

Common XML file keys to edit

<serialPortSettings> for C3, the default serial port is /dev/ttyLX4.

By default it is set to communicate with a baudrate of 9600, parity NONE, 1 stopbit, characterSize 8, and flowControl NONE. Ensure that the serial connector cables are plugged into the specified serial port on the SMC.

<powerOnVerb> and **<powerOffVerb>** for C3, the command string references power switch. Ensure that the power cable is plugged into the specified power switch on the SMC.

<defaultOutfile> for storing all header and data strings received from the C3, on the SMC filesystem. The default specified is c3data.txt in the local directory (/home/smc/roots/200_C3)

C3 is a command and event interface to a Turner C3 fluorometer; this is a re-implementation of the PIB-based C3 interface, and the commands are kept similar for continuity.

ConnectionManager Configuration

The ConnectionManager module manages various communication transports, such as a Cell Modem. It listens for requests for a channel (sorted by protocolRevision). It powers up a channel when the first request for a channel is received, and it powers down a channel when the last request is canceled. Additionally, ConnectionManager can fire events with the reported signal strength of a channel.

For Connection Manager to function correctly, the appropriate communication transports (eg, Cell Modem) must be wired up to the Ethernet port and to power switches appropriately.

The ConnectionManager application is started from

~ /roots/150_ConnectionManager/

And is configured by the XML file:

/roots/150_ConnectionManager/ConnectionManager-config.xml

ConnectionManager is assigned the board ID 32 (0x20) and the task ID 169 (0xA9).

Common XML file keys to edit

A sample configuration for the controller settings is as follows:

```
<channelSettings class_id="2" tracking_level="0" version="0">
    <count>2</count>
    <item_version>0</item_version>
    <item>
        <protocolRevision>32</protocolRevision>
        <powerOnVerb>./WiFiPowerOn</powerOnVerb>
        <powerOffVerb>./WiFiPowerOff</powerOffVerb>
        <signalStrengthVerb></signalStrengthVerb>
```

```

        <signalStrengthFilename></signalStrengthFilename>
    </item>
    <item>
        <protocolRevision>33</protocolRevision>
        <powerOnVerb>./CellPowerOn</powerOnVerb>
        <powerOffVerb>./CellPowerOff</powerOffVerb>
        <signalStrengthVerb>./GetSignalStrength_DigiConnect</signalStrengthVerb>
        <signalStrengthFilename>SignalStrength.txt</signalStrengthFilename>
    </item>
</channelSettings>

```

- **count**: This is the number of channels that the ConnectionManager will manage.
- **protocolRevision**: The protocolRevision of the channel. This needs to be unique for everything channel. This is how other processes identify which channel they are requesting.
- **powerOnVerb**: This is the command or script that is executed to turn on the channel.
- **powerOffVerb**: This is the command or script that is executed to turn off the channel.
- **signalStrengthVerb**: This is the command or script executed to retrieve the signal strength of the channel. If left blank, no signal strength will be reported.
- **signalStrengthFilename**: If the signalStrengthVerb is specified, this is the file where the signal strength of the channel is stored.

DataRecorder Configuration

The DataRecorder application is a module capable of archiving data received, over a serial port, in a data file on the SMC filesystem and making it available for extraction at runtime. The DataRecorder application also provides an interface for switching a pre-specified power switch (that the data sending device may be connected to), changing the communications settings of the serial port, changing the archive data file, and pause-resume recording of the data being received over the serial port. There are two instances of the DataRecorder installed on the SMC, designed to be used independently using separate serial ports and power switches. These are setup and configured in:

```
~/roots/200_DataRecorder1 and ~/roots/200_DataRecorder2
```

DataRecorder1 instance is configured with the serverName **DataRecorder1**, and assigned the boardId 32 (hexadecimal 20) and taskId 48 (hexadecimal 30).

DataRecorder2 instance is configured with the serverName **DataRecorder2**, and assigned the boardId 32 (hexadecimal 20) and taskId 49 (hexadecimal 31)..

Common XML file keys to edit

- **serialPortSettings** for DataRecorder1, the default serial port is /dev/ttyLX6 and for DataRecorder2 the default serial port is /dev/ttyLX7. By default, both are set to communicate with a baudrate of 9600, parity NONE, 1 stopbit,

characterSize 8, and flowControl NONE. Ensure that the serial connector cables are plugged into the specified serial ports on the SMC.

- **outFileName** for DataRecorder1, the default archive file to record data received from the serial port is, **recorder1.out** in the default directory ~/roots/200_DataRecorder1. For DataRecorder2, the default archive file to record data received from the serial port is, **recorder2.out** in the default directory ~/roots/200_DataRecorder2.
- **powerOnVerb** and **powerOffVerb** for DataRecorder1, the command string references power switch 3 and for DataRecorder2 the command string references power switch 4. Ensure that the power cables are plugged into the specified power switches on the SMC.

Using the DataRecorder

The DataRecorder application supports the following commands (sent on the SMAC via ServerExec using the serverName of the target instance, or from shore shade applications by specifying the boardId and taskId of the target instance):

START

setup recorder in resume (recording) mode and turn on the power switch.

STOP

turn off the power and set the recording mode to pause.

PAUSE

pause recording of data being received from the serial port (read & discard)

RESUME

continue recording data received from the serial port into the active output file.

OUTPUT <filename>

record subsequent data to the file <filename>.

ON

set power switch to ON

OFF

set power switch to OFF

SETPORT <baudrate> [<parity> [<characterSize> [<stopBits>]]]

change serial port's settings.

The above commands themselves are not case sensitive, however arguments such as <filename> are used as case-sensitive names.

Example: Output /mnt/XDATA/recorder1data.out
 OUTPUT /mnt/XDATA/Archive1.TXT
 output /home/smc/roots/200_DataRecorder2/rec2.bin

GPSListener Configuration

GPSListener is a service that listens to the NMEA sentences being received from the navigation computer, typically on /dev/ttyS1 or /dev/ttyO1 and:

- Parses the sentences to formulate GPS events
- Transmits GPS events over CORBA
- Sets the system time in based on the content of the event, using the PPS (pulse per second) input to synchronize the timing

Commands

GPSListener supports the following commands:

GPS AUTOSTART [<state>]

If the *state* variable is provided, the service is configured to automatically begin storing data to the output file on startup ("true") or is left waiting for a START command ("false")

If the *state* variable is not provided, the current state of the AUTOSTART feature is returned from the command.

GPS DEFAULTOUTPUT [<filename>]

If the *filename* variable is provided, the default output file name is set to the given value; at startup, the output file will be set to this default file name.

If the *filename* variable is not provided, the default output file is returned from the command.

GPS OUTPUT [<filename>]

If the *filename* variable is provided, the current output file name is set to the given value.

If the *filename* variable is not provided, the current output file name is returned from the command.

GPS SIMULATE OFF

Turns off GPS simulation.

**GPS SIMULATE ON <center-latitude> <center-longitude> <speed-kt>
<radius-m> <CW|CCW>**

Starts GPS point generation using the given variables, assuming that the vehicle is traveling in a circle.

GPS START

Commands the service to begin storing received data into the output file. Note that if the AUTOSTART feature is enabled, storage will be automatically started at system startup.

GPS STOP

Commands the service to stop storing received data into the output file.

CORBA Events

GPSListener sends out GPS CORBA events according to the parameters in the XML file. The maximum receive rate for GPS data is 1Hz, but the XML file can specify events to be sent out at different rates by dividing the seconds down. The typical configuration is as follows:

```
<gpsEventSpecs class_id="3" tracking_level="0" version="0">
  <count>2</count>
  <item_version>1</item_version>
  <item class_id="4" tracking_level="0" version="1">
    <divisor>5</divisor>
    <divisorMode>0</divisorMode>
    <eventSettings class_id="5" tracking_level="0" version="0">
      <eventName>GPSSample</eventName>
      <eventType class_id="6" tracking_level="0" version="0">
        <typeName>GPSSample</typeName>
        <domainName>LiquidR-SMC</domainName>
      </eventType>
    </eventSettings>
  </item>
  <item>
    <divisor>1</divisor>
    <divisorMode>0</divisorMode>
    <eventSettings>
      <eventName>GPSSample1Sec</eventName>
      <eventType>
        <typeName>GPSSample</typeName>
        <domainName>LiquidR-SMC</domainName>
      </eventType>
    </eventSettings>
  </item>
</gpsEventSpecs>
```

In this configuration, there are two events sent out:

- GPSSample is sent out once every 5 seconds
- GPSSample1Sec is sent out once every 1 second

Both are of the same type (typename "GPSSample" and domain name "LiquidR-SMC") By convention the type name is used to indicate which class parses the CORBA data, so EventLib's GPSSample class models CORBA property names:

Field Name	Type	Description
GPS-DegreesMagnetic	DOUBLE FLOAT	Course direction in degrees magnetic (from VTG)

Field Name	Type	Description
GPS-DegreesTrue	DOUBLE FLOAT	Course direction in degrees true (from VTG)
GPS-Elevation	DOUBLE FLOAT	Antenna altitude (meters, from GGA)
GPS-Latitude	DOUBLE FLOAT	Position/latitude in degrees (from GGA)
GPS-Longitude	DOUBLE FLOAT	Position/longitude in degrees (from GGA)
GPS-MagneticDeclination	DOUBLE FLOAT	Magnetic declinations (from RMC)
GPS-NMEASentences	STRING ARRAY	The collection of NMEA sentences comprised in this sample
GPS-NumSatellites	UINT32	The number of satellites seen (from GGA)
GPS-RMSAccuracy	DOUBLE FLOAT	RMS (root mean square) of the standard deviations of latitude, longitude and altitude (components from RMC)
GPS-Simulated	BOOLEAN	True if this input was simulated, False if it came from an operating GPS receiver
GPS-SpeedKmh	DOUBLE FLOAT	Speed through course in km/h (from VTG)
GPS-SpeedKnots	DOUBLE FLOAT	Speed through course in knots (from VTG)
GPS-TimeSec	UINT32	UTC time in seconds since 1-Jan 1970 (from ZDA)
GPS-TimeUSec	UINT32	Microseconds component of time (from ZDA)

XML Configuration

The configuration file (GPSListener-config.xml), which must be located in the startup directory (typically /home/smc/roots/200_GPSListener), conforms to the format of other configuration files, but here are a few areas of interest:

```
<serialPortSettings class_id="2" tracking_level="0" version="0">
  <portName>/dev/tty01</portName>
  <baudRate>4800</baudRate>
  <parity>NONE</parity>
  <characterSize>8</characterSize>
  <stopBits>1</stopBits>
  <flowControl>NONE</flowControl>
  <rts>1</rts>
  <dtr>1</dtr>
  <breakTimeoutMultiplier>1</breakTimeoutMultiplier>
</serialPortSettings>
<maxString>1000</maxString>
<readTimeoutMs>50</readTimeoutMs>
```

Note that the serial port and the baud rate are configurable. *maxString* is the size of the read to be made: be sure that the entire set of strings can fit in that number of characters. *readTimeoutMs* is the inter-byte gap in milliseconds that triggers the data to be returned.

```
<dateSetDirectly>1</dateSetDirectly>
<dateVerb>sudo date --utc {mm}{dd}{HH}{MM}{yyyy}.{SS}</dateVerb>
<hwClockVerb>sudo hwclock -w</hwClockVerb>
<defaultOutputFileName>/mnt/XDATA/GPS/sensor.out</defaultOutputFileName>
```

dateSetDirectly should be non-zero if using the Linux system API's to set the clock; that is the most efficient way. If not setting the clock using the system API's, the *dateVerb* can be used to set the time using **date**, but this will result in additional jitter.

Also note that *hwClockVerb* is provided to copy transfer the system date/time to the hardware clock. An accurate hardware clock is useful to set the system time before GPS is received.

```
<deltaToSetTimeMs>100</deltaToSetTimeMs>
<deltaToSetHWClockMs>0</deltaToSetHWClockMs>
```

The system time is not updated unless there is a difference of at least *deltaToSetTimeMs* milliseconds; if it is set, the hardware clock is update if it has not been set before or if at least *deltaToSetHWClockMs* milliseconds has elapsed since the last setting. Note that if *deltaToSetHWClockMs* is zero, the hardware clock is only ever set once.

```
<rmsAccuracyThreshold>50</rmsAccuracyThreshold>
```

If the RMS calculation considering the standard deviation of latitude, longitude and altitude exceeds *rmsAccuracyThreshold*, the GPS sample is not sent.

```
<gpsEventSpecs class_id="3" tracking_level="0" version="0">
  <count>2</count>
  <item_version>1</item_version>
  <item class_id="4" tracking_level="0" version="1">
    <divisor>5</divisor>
    <divisorMode>0</divisorMode>
    <eventSettings class_id="5" tracking_level="0" version="0">
      <eventName>GPSSample</eventName>
      <eventType class_id="6" tracking_level="0" version="0">
        <typeName>GPSSample</typeName>
        <domainName>LiquidR-SMC</domainName>
      </eventType>
    </eventSettings>
  </item>
  <item>
    <divisor>1</divisor>
    <divisorMode>0</divisorMode>
    <eventSettings>
      <eventName>GPSSample1Sec</eventName>
      <eventType>
        <typeName>GPSSample</typeName>
        <domainName>LiquidR-SMC</domainName>
      </eventType>
    </eventSettings>
  </item>
</gpsEventSpecs>
```

This section configures how many events are sent out and on what frequency. In this case, there are two events being sent:

- “GPSSample” reports GPS on 5-second boundaries
- “GPSSample1Sec” reports GPS on 1-second boundaries

The number of events can be customized to reduce overhead or increase resolution.

```
<tryBaudRates>
  <count>2</count>
  <item_version>0</item_version>
  <item>4800</item>
  <item>9600</item>
</tryBaudRates>
```

This section determines which baud rates will be tried: SV2 is set up to report GPS at 4800 baud, while SV3 reports it at 9600 baud.

```
<setTimeOnPPS>true</setTimeOnPPS>
<ppsGPIO>167</ppsGPIO>
```

If *setTimeOnPPS* is true, the system date/time will only be updated on the edge of the PPS (pulse per second) output of the GPS receiver. The signal being monitored is configured by *ppsGPIO*

Logger

The Logger utility can be used in place of **tee** to write log files from process output, and it allows limitations to be placed on the size of log files produced. The following help is available by issuing **Logger --help**:

```
Logger: Logger { ... options ... }
```

Options can be specified as a collection of single-letter shortcuts preceeded by a dash and followed by tokens to use as arguments or as long-form options preceeded by double-dashes with arguments formatted as assignments. For example:

```
Logger -wc /my-partition/workfiles BZIP2 --prefix=MYLOG --  
extension=.EXT --max-files 10 --max-file-size 1000000
```

The 'w' and 'c' short form options are assigned '/my-partition/workfiles' and 'BZIP2'; the --prefix option is assigned 'MYLOG'; the --extension option is assigned '.EXT'; the --max-files option is assigned '10'; and the --max-file-size option is assigned '1000000'

The available options are as follows:

-a, --archive-dir	The directory where log archives are stored (defaults to work directory)
-c, --archive-compression	The type of archive compression to use: BZIP2, GZIP or NONE
-e, --extension	The extension used for each output file (before compression)
-f, --max-files	The maximum number of files (archives+work file) to retain
-g, --purge-strategy	The strategy to use when purging files: OLDEST or NONE
-k, --keep-writes-together	Allow lines to break across files (0) or keep writes/lines together (1)
-p, --prefix	The first part of the file name for each file in the set
-q, --max-bytes-queued	The size of the inter-thread write queue in bytes (larger value delays the logging client less)
-r, --max-retain-time	The maximum number of seconds to retain archives (0=no limit)
-s, --max-file-size	The maximum size of each file (before compression)
-w, --work-dir	The directory for the work file and default for archive files; starting directory if not specified

GPSWaves Configuration (GPS and IMU based waves sensor)

Interface Microstrain 3DM-GX35 GPS and IMU sensor and use its GPS output to compute wave height parameters (using Jim Thomson's algorithms.) Keep the software that Jim Thomson produces in a separate library so that it can be substituted separately from the main program.

Commands

Commands to GPSWaves are sent to the process name "GPSWaves" with ServerExec or it is typically assigned an address of 0x2025 (the upper byte varies with the base address of the SMC.) The available commands are:

GWAVES --autostart [<new-state>]

If the *new-state* parameter is given, it sets the state of the automatic start feature. If *new-state* is **true**, the service starts sampling and returning data when the program launches; otherwise, the program remains idle after launch and an explicit start command must be sent to start sampling.

If the *new-state* parameter is omitted, the command returns the current state of the automatic start feature: **true** (begin sampling on program launch) or **false** (idle on program launch.)

GWAVES --period <minutes>

Specifies the period, in minutes, over which the Microstrain data samples are collected for processing by Jim Thomson's library, for calculating wave height and spectrum. **<minutes>** can be any value between 0 and 60, but values lower than 20 minutes are not recommended as that will result in insufficient data samples to perform wave height calculations. The system default is set at 30 minutes.

GWAVES --start

Powers on the Microstrain IMU and begins sampling, storing, and processing its data for wave height calculations.

GWAVES --stop

Stops sampling and sending wave height calculations and powers down the Microstrain.

CORBA Events

GPSWaves sends out one event (GPSWavesTelemEvent) to report the wave height calculations for each period (duration of which is specified as described above); it contains the following fields:

Field Name	Type	Description
Hs	DOUBLE FLOAT	Wave height (in meters)
Tp	DOUBLE FLOAT	Peak period (in seconds)
Dp	DOUBLE FLOAT	Dominant direction (in degrees)
Samples	UINT16	Number of samples used for this calculation
Fs	DOUBLE FLOAT	Frequency of data samples during this period's calculation
SampleGaps	UINT16	Number of missing/dropped samples during this period

The event is sent back to WGMS using the following entry in PayloadInterface-telemetry-config.xml:

```
<Element>
  <first boardID="0xFF" taskID="0x25" />
  <second>
    <key boardID="0xFF" taskID="0x25" />
    <payloadType value="0x00000093" />
  </second>
</Element>
```

```

        <eventNameToFormatCode>
            <Element>
                <first value="GPSWavesTelem" />
                <second value="0" />
            </Element>
        </eventNameToFormatCode>
        <reportParams>
            <Element>
                <first value="0" />
                <second maxSamples="1" targetAddress="2">
                    <reportHeaderFormat>
                        <fields>
                            <Element libField="Header91" />
                            <Element libField="Raw" eventField="Telem-
LengthToFollow" />
                            <Element libField="Raw" eventField="Telem-PayloadType"
/>
                            <Element libField="Raw" eventField="Telem-BoardID" />
                            <Element libField="Raw" eventField="Telem-TaskID" />
                            <Element libField="FC0BinaryNormal" />
                            <Element libField="RawINT8" eventField="Telem-
NumSamples" />
                        </fields>
                    </reportHeaderFormat>
                    <sampleHeaderFormat>
                        <fields>
                            <Element libField="Raw" eventField="Telem-Latitude" />
                            <Element libField="Raw" eventField="Telem-Longitude" />
                            <Element libField="Raw" eventField="Telem-TimeSec" />
                            <Element libField="Raw"
eventField="GPSWavesTelemEvent_Hs" />
                            <Element libField="Raw"
eventField="GPSWavesTelemEvent_Tp" />
                            <Element libField="Raw"
eventField="GPSWavesTelemEvent_Dp" />
                            <Element libField="Raw"
eventField="GPSWavesTelemEvent_Samples" />
                            <Element libField="Raw"
eventField="GPSWavesTelemEvent_Fs" />
                            <Element libField="Raw"
eventField="GPSWavesTelemEvent_SampleGaps" />
                        </fields>
                    </sampleHeaderFormat>
                </second>
            </Element>
        </reportParams>
    </second>
</Element>

```

XML Configuration

The configuration file (GPSWaves-config.xml), which must be located in the startup directory (typically /home/smc/roots/200_GPSWaves), conforms to the format of other configuration files, but the following items are of particular interest:

```

<controllerSettings class_id="12" tracking_level="0" version="1">
    <serialPortSettings class_id="13" tracking_level="0" version="0">
        <portName>/dev/ttyLX4</portName>
        <baudRate>115200</baudRate>
        <parity>NONE</parity>
        <characterSize>8</characterSize>
        <stopBits>2</stopBits>
        <flowControl>NONE</flowControl>
        <rts>1</rts>
    </serialPortSettings>
</controllerSettings>

```

```

        <dtr>1</dtr>
        <breakTimeoutMultiplier>1</breakTimeoutMultiplier>
    </serialPortSettings>
    <powerOnVerb>sudo SetPowerSwitch 2 1</powerOnVerb>
    <powerOffVerb>sudo SetPowerSwitch 2 0</powerOffVerb>
    <outputFileName>/mnt/XDATA/GPSWaves/rawdata.txt</outputFileName>

Name>    <processedDataFileName>/mnt/XDATA/GPSWaves/GPSwaves.txt</processedDataFile

        <spectrumFileName>/mnt/XDATA/GPSWaves/spectrum.txt</spectrumFileName>
        <samplingFrequency1000>4000</samplingFrequency1000>
        <seaStateInterval>30</seaStateInterval>
        <lfnr>2</lfnr>
        <nStdDev>5</nStdDev>
        <enableRawDataEvent>false</enableRawDataEvent>
        <enableSpectrumEvent>false</enableSpectrumEvent>
        <enableTelemEvent>true</enableTelemEvent>
    </controllerSettings>

```

The **samplingFrequency1000** must match the Microstrain's GPS data frequency configuration and is the frequency*1000

LuaWrapper Configuration

Commands

The following commands can be sent to LuaWrapper via ServerExec or by targeting its address with Write Console Script:

LWM -list

Lists the lua instance names and the file that was loaded into each instance.

LWM --loadlua *lua-script-file instance-name*

Creates a new lua instance named for *instance-name* and initializes with the script file indicated by *lua-script-file*

LWM --unloadlua *instance-name*

Unloads the lua instance previously-loaded as *instance-name*

LWM --run *command-file*

Loads the text lines from *command-file* and executes them as a set of LuaWrapper commands.

LWSCRIPT --invoke *instance-name function-name* { ...*argument-list*... }

Invokes the function named *function-name* on the script that is running in the previously-loaded instance, *instance-name*

Special Variables

LuaWrapper makes the following variables and meta-tables available to the script:

- SMC, which exposes the primary API to the script
- SMCContext, which exposes an invocation-specific API to the script to control when the **LWSCRIPT --invoke** command returns
- A serial port meta-table that is returned by SMC:NewSerialPort
- A timer meta-table that is returned by SMC:NewTimer

The SMC meta-table is as follows:

gps-reading = SMC:GetLocation()

Returns the last GPS reading as a table with elements "latitude" and "longitude" set to the whole and fractional degrees; if no GPS reading has been received, GetLocation returns nil. For example:

```
local lastReading = SMC:GetLocation()
if lastReading then
    print("Last GPS reading: ("
        ..tostring(lastReading.latitude)..", "
        ..tostring(lastReading.longitude)..")")
end
```

SMC:Log(*log-level*, *message*)

Outputs the *message* string to the process log at a priority (*log-level*) which may be any of: "debug", "info", "notice", "warning", "error", "critical", "alert" or "emergency"

timsp-var = SMC:NewSerialPort(*config-ID*)

Creates an instance of a serial port object based on the selected configuration (*config-ID*, a string.) The returned data type is a serial port meta-table (see below) Received data is posted to the lua script via the **SMCReportSerialData** function, if it is defined.

timer-var = SMC:NewTimer(*timer-ID*, *duration*)

Creates an instance of a timer object with the given ID (*timer-ID*, a string) and *duration* (a number of milliseconds.) If the timer is not deleted before the timeout expires, the script's **SMCReportTimeout** function is invoked, if it is defined.

SMC:PublishDataSet(*event-name*, *event-type*, *event-domain*, *data-set*, *content*)

Publishes a data set event with event characteristics set by *event-name*, *event-type* and *event-domain* (all strings.) In addition, the data set name (*data-set*, a string) and the *content* of the data set (a string) are provided.

<i>event-name</i>	<i>event-type</i>	<i>event-domain</i>	<i>destination</i>
PPPHostNewFileAvailable	DataSetEvent	LiquidR-SMC	WGMS via Iridium SBD
PyIdNewFileAvailable	DataSetEvent	LiquidR-SMC	WGMS via Iridium SBD

uptime-seconds = SMC:SystemUptime()

Returns the number of seconds since the SMC was started as a floating point number with a resolution of 0.01 seconds.

SMC:PublishEvent(*event-table*)

Uses *event-table* to create a CORBA event and publish it to EventManager. See the CORBA Event Table section, below.

The SMCContext meta-table is as follows:

SMCContext:AppendOutput(*data*)

Appends *data* string to the output that will be reported from the command.

SMCContext:ReportComplete()

For a command that was previously reported as asynchronous (SMCContext:SetAsynch), this indicates that the command is now complete.

SMCContext:SetAsynch()

Indicates that the invocation (LWSCSCRIPT --invoke) corresponding to this context variable should not return until ReportComplete() is called. This allows a more complex state machine to be implemented as a result of the lua script function being called.

SMCContext:SetSuccessFlag(*status*)

Sets or clears the success flag according the boolean variable "status" This value is carried back in the command completion event for use by the caller. Note that if an exception is thrown, command is considered to be unsuccessful; if SetSuccessFlag is not called and no exception is thrown, the command is considered to be successful.

The serial port meta-table (returned from SMC:NewSerialPort) is as follows:

ID = *sp-var*:GetID()

Returns the serial port configuration ID string associated with this serial port.

sp-var:SetBreak()

Puts the serial port in break mode (continually transmitting electrical low) for the default period of time.

sp-var:SetBreakTime(ms)

Puts the serial port in break mode (continually transmitting electrical low) for the specified number of milliseconds (ms.)

sp-var:Transmit(*data*)

Transmits the *data* string over the serial port.

The timer meta-table (returned from SMC:NewTimer) is as follows:

ID = *timer-var*:GetID()

Returns the timer ID associated with the timer.

CORBA Event Tables and Variable Definitions

CORBA events are used to report significant occurrences and their data to other interested processes via the EventManager interface. Each event has:

- A **name**, which indicates what has occurred.
- A **type**, consisting of a **type name** and a **domain name**; these help to categorize the event report. By convention, the SMC software uses the event **type** to define how the accompanying data is formatted.
- A series of **name/value** pairs that carry data relative to the event report.

Each of the fields above is a string with the exception of the **value**; each value can take on one of the following CORBA types:

CORBA Type ID	Meaning	Example(s) in lua
BOOLEAN	Boolean value or flag	true or false
BOOLEAN_VECTOR	array of Boolean values or flags	{ { true, false, false, true } }
DOUBLE_FLOAT	double-precision floating point	<i>number</i> or 3.141592654
DOUBLE_FLOAT_VECTOR	array of double-precision floating point	{ { 1, 2, 3, 4 } }
INT16	16-bit signed integer	{ <i>number</i> , "INT16" } or { 8193, "INT16" }
INT16_VECTOR	array of 16-bit signed integers	{ { 1, 2, 3, 4 }, "INT16_VECTOR" }
INT32	32-bit signed integer	{ <i>number</i> , "INT32" } or { -1000000, "INT32" }
INT32_VECTOR	array of 32-bit signed integers	{ { 1, 2, 3, 4 }, "INT32_VECTOR" }
INT8	8-bit signed integer	{ <i>number</i> , "INT8" } or { 5, "INT8" } or { -5, "INT8" }
INT8_VECTOR	array of 8-bit signed integers	{ { 1, 2, 3, 4 }, "INT8_VECTOR" }
NULL	nil	Nil
SINGLE_FLOAT	single-precision floating point	{ <i>number</i> , "SINGLE_FLOAT" } or { 3.14, "SINGLE_FLOAT" }
SINGLE_FLOAT_VECTOR	array of single-precision floating point	{ { 1, 2, 3, 4 }, "SINGLE_VECTOR" }
STRING	character string	"Hello, world" or { 12, "STRING" }
STRING_VECTOR	array of character strings	{ { "Mercury", "Venus", "Earth", "Mars" } }
UINT16	16-bit unsigned integer	{ <i>number</i> , "UINT16" } or { 40000, "UINT16" }

CORBA Type ID	Meaning	Example(s) in lua
UINT16_VECTOR	array of 16-bit unsigned integers	{ { 1, 2, 3, 4 }, "UINT16_VECTOR" }
UINT32	32-bit unsigned integer	{ <i>number</i> , "UINT32" } or { 1000000, "UINT32" }
UINT32_VECTOR	array of 32-bit unsigned integers	{ { 1, 2, 3, 4 }, "UINT32_VECTOR" }
UINT8	8-bit unsigned integer	{ <i>number</i> , "UINT8" } or { 12, "UINT8" }
UINT8_VECTOR	array of 8-bit unsigned integers	{ { 1, 2, 3, 4 }, "UINT8_VECTOR" }

The generic form of a CORBA event in lua table format is:

```
{ header =      { name = "event-name",
                  typename = "event-type-name",
                  domainname = "event-domain-name" },
  properties = { prop1name = prop1value,
                  prop2name = prop2value,
                  ...
                  propnname = propnvalue } }
```

For example, to publish an event:

```
local evt = { }

evt.header = { name = "LWTelem1",
               typename = "LWTelem1",
               domainname="LqrSMAC" }

evt.properties = { }
evt.properties["Telem-BoardID"] = { 32, "UINT8" }
evt.properties["Telem-TaskID"] = { 64, "UINT8" }
evt.properties["StringMessage"] = "Hello, world!"

SMC:PublishEvent(evt)
```

To receive an event and print out the fields to the console:

```
function SMCReportEvent(eventRx)

    print("SMCReportEvent name="..tostring(eventRx.header.name))
    print("header " ..tostring(eventRx.header))
    print("header.name " ..tostring(eventRx.header.name))
    print("header.typename " ..tostring(eventRx.header.typename))
    print("header.domainname " ..tostring(eventRx.header.domainname))

    print("properties " ..tostring(eventRx.properties))

    for k,v in pairs(eventRx.properties)
    do
        if type(v[1]) == "table" then
            local s2 = ""

            for k2,v2 in ipairs(v[1]) do
                if #s2 ~= 0 then
                    s2 = s2..","
                end
                s2 = s2..tostring(k2)..":"..tostring(v2)
            end

            print("properties["..k.."] ->
                "..type(v[1]).."/"..s2.." " ..tostring(v[2]))
        end
    end
end
```

```
        else
            print("properties[..k.."] ->
"..type(v[1]).."/"..tostring(v[1]).." "..tostring(v[2]))
        end
    end
end
```

Here's a simplified way of handling the typical GPS distribution event as a generic event:

```
function SMCReportEvent(eventRx)
    if tostring(eventRx.header.name) == "GPSSample" then
        local latitude = eventRx.properties["GPS-Latitude"][1]
        local longitude = eventRx.properties["GPS-Longitude"][1]

        if latitude and longitude then
            print("gps latitude="..tostring(latitude)..
longitude="..tostring(longitude))
        end
    end
end
```

Sometimes string data will be sent out as an array of bytes (UINT8_VECTOR.) Byte arrays can be converted to string as follows:

```
local field = eventRx.properties["some-property"]
local data = field[1][1]

if type(data)=="table" and tostring(field[2][2]) == "UINT8_VECTOR" then
    data = string.char(unpack(data))
end
```

Pre-defined Callback Functions

LuaWrapper reserves the following script functions for call-backs:

```
function SMCReportEvent(eventRx)
```

This function is called whenever an event is received and is successfully converted to a lua table (see CORBA Event Tables and Variable Definitions, above.) Note that if the Event Transceiver settings in the XML have enabled the event filter and the received event is filtered out, SMCReportEvent not called.

```
function SMCReportSerialData(ID,data)
```

This function is called whenever serial data is received according to the configuration specified in the XML and referenced when opening the port:

ID is a string matching the configuration name passed to SMC:NewSerialPort;

data is a string containing the received data.

```
function SMCReportTimeout(ID)
```

This function is called whenever a timer expires:

ID is a string matching the timer ID passed to SMC:NewTimer

XML Configuration

The configuration file (LuaWrapper-config.xml), which must be located in the startup directory (typically /home/smc/roots/200_LuaWrapper), conforms to the format of other configuration files, but the controllerSettings section is of particular interest:

```
<?xml version="1.0"?>
<controllerSettings class_id="13" tracking_level="0" version="1">
  <serialConfigMap class_id="14" tracking_level="0" version="0">
    <count>1</count>
    <item_version>0</item_version>
    <item class_id="15" tracking_level="0" version="0">
      <first>config1</first>
      <second class_id="16" tracking_level="0" version="1">
        <listenerSettings class_id="17" tracking_level="0" version="0">
          <maxReadSize>4096</maxReadSize>
          <intercharacterGapTimeoutMs>50</intercharacterGapTimeoutMs>
        </listenerSettings>
        <portSettings class_id="18" tracking_level="0" version="0">
          <portName>/dev/ttyLX2</portName>
          <baudRate>9600</baudRate>
          <parity>NONE</parity>
          <characterSize>8</characterSize>
          <stopBits>1</stopBits>
          <flowControl>NONE</flowControl>
          <rts>1</rts>
          <dtr>1</dtr>
          <breakTimeoutMultiplier>1</breakTimeoutMultiplier>
        </portSettings>
      </second>
    </item>
  </serialConfigMap>
</controllerSettings>
```

The **serialConfigMap** is a dictionary of settings (**second** section) for serial ports keyed by configuration name (**first** section); the configuration name is referenced by the SMC:NewSerialPort() function when opening a serial port. Various named configurations can be created in the XML file by repeating the **item** section and varying the configuration name. Note that **count** must be set to the number of **item** sections that will appear within the **serialConfigMap**.

Sample lua script

The following is a test case that uses some of the features above. Although the scripting interface is necessarily event-oriented to keep it single-threaded, this script shows how lua coroutines can be used to implement a top-down application that sends data to a serial port and listens for responses.

```
-- "Top-down" application module using co-routines to execute
-- sporadically on I/O and timing events

function Application()

    SMC:Log("debug", "Starting Application")

    port = SMC:NewSerialPort("config1")

    local firstTime, thisTime, delta, waitTime

    thisTime = SMC:SystemUptime()
    firstTime = thisTime-1
```

```

while true
do
    delta = thisTime-firstTime
    waitTime = delta-math.floor(delta)

    if (waitTime >= 0) and (waitTime < 1) then
        waitTime = math.ceil(1000*(1-waitTime))
    else
        waitTime = 1000
    end

    SMC:Log("debug", "Waiting "..tostring(waitTime))
    Wait(waitTime)
    SMC:Log("debug", "Back from waiting")

    local i
    local awaitResult

    i = 1
    awaitResult = false

    while (i <= 3) and (awaitResult == false)
    do
        port:Transmit("AT\r")
        awaitResult = Await("OK\r\n", 300)
        i = i+1
    end

    thisTime = SMC:SystemUptime()

end

end

-- Waits for the given serial sequence to be received within the given
-- number of milliseconds; returns true if successful.
function Await(sequence,timeout)

    SMC:Log("debug", "Await sequence="..sequence.." timeout="..timeout)

    local rxd, result

    awaitTimer = SMC:NewTimer("await", timeout)
    result = false

    while (awaitTimer ~= nil) and (result == false)
    do
        rxd = coroutine.yield()
        if (rxd == sequence) then result = true end
    end

    SMC:Log("debug", "Await rxd="..rxd.." at loop exit")

    awaitTimer = nil
    collectgarbage("collect")

    SMC:Log("debug", "Await:"..tostring(result))
    return result

end

-- Delays for the given number of milliseconds
function Wait(timeout)

    waitTimer = SMC:NewTimer("wait", timeout)

    while waitTimer ~= nil
    do

```

```

        coroutine.yield()
    end
end
-- Callback routine for serial data received
function SMCReportSerialData(portName,data)
    SMC:Log("debug", "Recieved "..data.." on "..portName)
    if awaitTimer ~= nil then
        awaitTimer = nil
        collectgarbage("collect")
        coroutine.resume(appRtn, data)
    end
end
-- Callback routine for timer expirations
function SMCReportTimeout(timerName)
    SMC:Log("debug", "SMCReportTimeout timerName="..timerName)
    if timerName == "wait" then
        waitTimer = nil
        collectgarbage("collect")
        coroutine.resume(appRtn, nil)
    elseif timerName == "await" then
        awaitTimer = nil
        collectgarbage("collect")
        coroutine.resume(appRtn, nil)
    end
    SMC:Log("debug", "exiting SMCReportTimeout")
end
-- Initialize the module
SMC:Log("debug", "Starting test4")
waitTimer = nil
awaitTimer = nil
-- Launch the application on file load
appRtn = coroutine.create(function()
    Application()
end)
coroutine.resume(appRtn)
SMC:Log("debug", "test4 started")

```

Magnetometer Configuration (Marine Magnetics Explorer)

Magnetometer interfaces a Marine Magnetics Explorer and collects the data that it streams. A single sample is returned once per minute as telemetry and the interface provides the option to return logged data.

Commands

Commands to Magnetometer are sent to process name "Magnetometer" with ServerExec or it is typically assigned an address of 0x201D (the upper byte varies with the base address of the SMC.) The available commands are:

MAG --run <filename>

Sequentially executes the commands in the file of the given name (*filename*)

MAG --autostart [<new-state>]

If the *new-state* parameter is given, it sets the state of the automatic start feature. If *new-state* is true, the service starts sampling and returning data when the program launches; otherwise, the program remains idle after launch and an explicit start command must be sent to start sampling.

If the *new-state* parameter is omitted, the command returns the current state of the automatic start feature: true (begin sampling on program launch) or false (idle on program launch.)

MAG --getdata [--bzip2] <timestamp>

Retrieves the data received starting at the given timestamp, which must be in *yyyymmddHHMM* form:

yyyy = year
mm = month
dd = day of month
HH = hour
MM = minute

The entire minute of data is returned, up to 60 lines of text. If the **--bzip2** option is specified, the output is compressed before it is returned.

If compression is used and command is issued from WGMS, please use function code 1 so that the result is returned as a file. Note that function code 0 results to WGMS are limited to only a few messages, so it may be wise to use function code 1 in either case.

MAG --start

Powers up the Explorer and begins sampling and storing data.

MAG --stop

Stops sampling and powers down the Explorer.

MAG --syncclock

Immediately sets the clock on the Explorer to match the SMC's system time. Note that clock is also set when communication to the Explorer is started and at a configurable interval (see XML file description, below.)

MAG --zeropressure

Immediately zero's the pressure sensor reading at the current level; this is useful during deployment of the sensor to take out any offset resulting from shipping.

CORBA Events

Magnetometer sends out one event (MagnetometerRawData) to report the sample taken at zero seconds every minute; it just carries the literal string received from the Explorer:

Field Name	Type	Description
StringEvent-String	STRING	The literal message received from the Explorer at offset 0 in the minute

The event is sent back to WGMS using the following entry in PayloadInterface-telemetry-config.xml:

```
<?xml version="1.0"?>
<Element>
  <first boardID="0xFF" taskID="0x1D"/>
  <second>
    <key boardID="0xFF" taskID="0x1D"/>
    <payloadType value="0x0000008C"/>
    <eventNameToFormatCode>
      <Element>
        <first value="MagnetometerRawData"/>
        <second value="0"/>
      </Element>
    </eventNameToFormatCode>
    <reportParams>
      <Element>
        <first value="0"/>
        <second maxSamples="2" targetAddress="2">
          <reportHeaderFormat>
            <fields>
              <Element libField="Header91"/>
              <Element libField="Raw" eventField="TelemLengthToFollow"/>
              <Element libField="Raw" eventField="TelemPayloadType"/>
              <Element libField="Raw" eventField="TelemBoardID"/>
              <Element libField="Raw" eventField="TelemTaskID"/>
              <Element libField="FC0ASCIIINormal"/>
              <Element libField="RawINT8" eventField="TelemNumSamples"/>
            </fields>
          </reportHeaderFormat>
          <sampleHeaderFormat>
            <fields>
              <Element libField="Raw" eventField="TelemLatitude"/>
              <Element libField="Raw" eventField="TelemLongitude"/>
              <Element libField="Raw" eventField="TelemTimeSec"/>
              <Element libField="Raw" eventField="StringEventString"/>
            </fields>
          </sampleHeaderFormat>
        </second>
      </Element>
    </reportParams>
  </second>
</Element>
```

XML Configuration

The configuration file (Magnetometer-config.xml), which must be located in the startup directory (typically /home/smc/roots/200_Magnetometer), conforms to the format of other configuration files, but the following keys are of particular interest:

```
<clockUpdateInterval class_id="14" tracking_level="0" version="0">
  <is_special>0</is_special>
  <time_duration_hours>24</time_duration_hours>
  <time_duration_minutes>0</time_duration_minutes>
  <time_duration_seconds>0</time_duration_seconds>
  <time_duration_fractional_seconds>0</time_duration_fractional_seconds>
</clockUpdateInterval>
```

clockUpdateInterval configures the time interval between synchronizing the Explorer's clock to the SMC's system time.

```
<fileSetSettings class_id="18" tracking_level="0" version="0">
  <archiveCompressionType>BZIP2</archiveCompressionType>
  <archiveDirectory></archiveDirectory>
  <extension>.csv</extension>
  <keepWritesTogether>1</keepWritesTogether>
  <maxBytesQueued>10000</maxBytesQueued>
  <maxFileSize>1000000</maxFileSize>
  <maxFiles>1</maxFiles>
  <maxRetainTime>00:00:00</maxRetainTime>
  <prefix>MAGDATA</prefix>
  <purgeStrategy>NONE</purgeStrategy>
  <workingDirectory>/mnt/XDATA/Magnetometer/MAGDATA</workingDirectory>
</fileSetSettings>
```

fileSetSettings configures where and how the raw magnetometer readings are stored in the filesystem.

- Files are stored under “/mnt/XDATA/Magnetometer/MAGDATA” (*workingDirectory*)
- Files are named as MAGDATA*.csv (*prefix* and *extension*) and can be up to 1,000,000 bytes (*maxFileSize*); at that threshold, files are compressed using “bzip2” (*archiveCompressionType*; alternatives are “none” or “gzip”)
- There is no enforcement of the maximum number of files because *purgeStrategy* is set to “none”; the alternative is “oldest” which causes the oldest files to be removed when *maxFiles* or *maxRetainTime* is violated.
- There is no separate archive directory, but if one were specified, the renamed and compressed files would be moved there instead of staying in the working directory.

MAM Configuration (Datawell MOSE)

MAM (MOSE Access Manager) interface a Datawell MOSE motion sensor and implements algorithms for computing wave spectrum and related summary values. This section indicates what has changed and gives some setup guidance, but please consult the LRI Datawell Sensor User's Guide and the MOSE documentation for additional detail.

Commands

Commands to MAM are sent to process name “MOSESensor” with ServerExec or it is typically assigned an address of 0x2022 (the upper byte varies with the base address of the SMC.) The available commands are:

MOSE --getconfig

Reads the configuration from the sensor by issuing command
"\$PDTWL,CFG*"

MOSE --off

Turns off power to the sensor by executing the power off verb
from the XML file.

MOSE --on

Turns on power to the sensor by executing the power on verb
from the XML file.

MOSE -o *file-name*

Sets the name and path of the file to which raw samples are
logged.

MOSE --output *file-name*

This is a synonym for MOSE -o

1.1.6 MOSE --procout *file-name*

Sets the name and path of the file to which wave summary
data is logged; this includes wave height (hs), wave average
(tz), dominant wave period (tp) and wave direction (dirp.)

MOSE --reportrate *new-rate*

Configures the number of minutes between spectrum
calculations (1..60)

MOSE --reset

Resets the sensor by sending the "\$PDTWL,RST*" command.

MOSE --setconfig *LFSP LFLP HLFP SA-resist*

Sends configuration parameters to the sensor:

- **LFSP** is the low frequency, short period cut off: 1 = 10s, 2 = 30s
- **LFLP** is the low frequency, long period cut off: 1 = 300s, 2 = 600s, 3 = 1000s
- **HLFP** is high frequency, long period cut off: 1 = 30s, 2 = 60s, 3 = 100s
- **SA-resist**: 0 = off, 1 = ON

All four values may be specified, asterisks may be specified for
unchanged parameters, or trailing parameters may be omitted
if the values are unchanged.

1.1.6 MOSE --specout *file-name*

Sets the name and path of the file to which Datawell spectrum
datasets are logged.

1.1.6 MOSE --spec2out *file-name*

Sets the name and path of the file to which NDBC-compatible
wave spectrum data is logged.

MOSE --start

Begins logging data to files as configured.

MOSE --stop

Stops logging data to files.

MOSE --version

Reads the MOSE software version from the sensor by issuing the "\$PDTWL,VER*" command.

Common XML file keys to edit

- **controllerSettings** → **samplesPerSpectrum** is the count of samples to include in a spectrum sample; it is typically set to 256 and must be set to a power of 2.
- **controllerSettings** → **seaStateInterval** is the number of minutes between spectrum calculations; it is typically set to 60, so the spectrum measurement is taken once per hour.
- **controllerSettings** → **includeBadSamples** can be set to a non-zero value for debugging, but should be deployed set to zero (no bad samples are included in calculations.)
- **controllerSettings** → **stabilizedDuration** is the number of minutes to allow for GPS acquisition. MAM may turn the the sensor off to save power, but it must allow some startup time.
- **controllerSettings** → **collectingDuration** is the number of minutes to allow for collection of samples (sampleNum); this time in addition to stabilizedDuration is the minimum "on" time that MAM budgets for a spectrum collection.

PayloadInterface Configuration

PayloadInterface implements the SV2 Payload protocol or the SV3 REST API in order to communicate with the Float C&C and other devices on the Float and to interact with WGMS via file transfer and telemetry reports. Starting in revision 2.1.0, the user must select which protocol is used with the **sv3Enabled** key:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE boost_serialization>
<boost_serialization signature="serialization::archive" version="10">
<rootlevel class_id="0" tracking_level="0" version="2">
  <loggerSettings class_id="1" tracking_level="0" version="0">
    <logLevel>LT_INFO</logLevel>
    <processName>PayloadInterface</processName>
    <isFileDes>1</isFileDes>
    <fileDes>1</fileDes>
    <fileName></fileName>
    <pipeName></pipeName>
  </loggerSettings>
  <sv3Enabled>true</sv3Enabled>
  <serialPortSettings class_id="2" tracking_level="0" version="0">
```

The value is set to **true** if PayloadInterface is using the SV3 REST API or **false** if it's using the payload protocol.

PayloadInterface doesn't typically require any configuration, except to ensure that the required telemetry settings are merged into **PayloadInterface-telemetry-**

config.xml The shipping version contains the superset of the telemetry required by the core components; but if components are added, their telemetry formats need to be merged.

Starting in version 1.1.1, for situations where there is no direct GPS distribution (perhaps when the SMC is located in the SV2 sub), set **distributeGPS** to 1 and enter a command to set the date under **dateVerb** (sudo date -utc {mm}{dd}{HH}{MM}{yyyy}.{SS})

In an SV3 configuration, it is possible to receive GPS distribution via the umbilical aux lines or to receive GPS updates every second via the SV3 REST API. PayloadInterface automatically switches to reading GPS via the SV3 REST API when no NMEA stream is detected.

PayloadInterface Commands

PayloadInterface implements the following commands, regardless of the communication protocol:

GETMODULES

Returns a table of the configured CORBA servers and their board addresses. For example:

```
$ ServerExec PayloadInterface GETMODULES
Address  Readable Name
-----  -
0x2000   PayloadInterface
0x20F1   GPSListener
```

```
My board address = 0x2000
Default board ID = 0x20
```

RUNSHELL *command-line*

Runs *command-line* as a Linux shell command and returns the output as a command result. For example:

```
$ ServerExec PayloadInterface 'RUNSHELL ls -l'
total 84
-rw-rw-r-- 1 smc smc 13922 May  9 23:13 PayloadInterface-config.xml
-rw-rw-r-- 1 smc smc  8881 May  9 02:29 PayloadInterface-telemetry-config.xml
-rw----- 1 smc smc   968 May 13 23:15 PayloadInterface.json
-rwx----- 1 smc smc   202 Apr 30 21:49 startup
```

SendAlarmCleared sender=<address> text=<text> {reporter=<ID>}
{notification=<ID>} {enum=<ID>} {data=<hex string>}
SendAlarmSet sender=<address> text=<text> {reporter=<ID>}
{notification=<ID>} {enum=<ID>} {data=<hex string>}
SendAlarmEvent sender=<address> text=<text> {reporter=<ID>}
{notification=<ID>} {enum=<ID>} {data=<hex string>}
SendEvent sender=<address> text=<text> {reporter=<ID>} {notification=<ID>}
{enum=<ID>} {data=<hex string>}

These four commands allow different kinds of alarm events to be sent from the payload. The commands were included for testing: there is a CORBA message (AlarmMessage) that carries this same information; however, it is possible to implement the interface using ServerExec and these commands:

- SendAlarmSet reports an alarm in a persistent state: it cannot be cleared by the operator as long as it persists.
- SendAlarmCleared reports that a previously-reported persistent alarm is no longer asserted, and it can be completely cleared.
- SendAlarmEvent reports an alarm as a one-time event: the operator can clear it right away.
- SendEvent reports a non-alarm event.

The parameters are as follows:

- The sender address is the 16-bit board ID listed as the source of the message (required)
- The text field is the free-form text to report (required)
- The reporter ID is a 32-bit ID number that identifies a type of reporter (optional, 0 by default)
- The notification ID is a 32-bit number that identifies a kind of alarm associated with the reporter (optional, 0 by default)
- The enumerated ID is a 32-bit number that is used in combination with reporter ID and notification ID to make alarms that are set and cleared unique (optional, 0 by default)
- By convention, reporter ID values 1-999 are reserved to LRI and values 1000-1999 are reserved LRI-developed SMC software. Third-party developers should contact LRI to obtain a reserved range.

SMC services are setting the enumerated ID equal to the board address of the service, which allows alarms from difference service instances to be differentiated. For example, if there are multiple ADCP's in the same payload box, or if there are ADCP's in two different payload boxes, their alarms will not be conflated.

PayloadInterface implements the following commands only when running the SV2 payload protocol:

EXECFCC *hex-string*

Decodes *hex-string* as a sequence of bytes and re-interprets it as a Level 2 message. A CRC is added and the result is sent to the Float C&C as a command. This is a test function.

FLOATREBOOT

Sends a Float reboot message to the Float C&C, which causes the Float C&C software to restart.

LIGHTOFFTEST

Sends a “light off” command to the Float C&C, which causes the Float C&C to turn off its visibility light. This is a test function.

SETPARAM *parameter-name=new-value { parameter-name₂=new-value₂ { ... } }*

Sends a “Set Parameters” message, with one or more parameters set to new values. The following are the parameters that can be set:

Parameter Name(s)	Description	Type
AIS	AIS Power	Boolean (true or false)
Bubble	Bubble Board Power	Boolean (true or false)
CamCtrl	Camera Control	Boolean (true or false)
ENUMDELAY	Payload power-on enumeration delay (seconds)	16-bit unsigned integer
FASTMB	Fast Mailbox check enabled (forces Iridium session every minute)	Boolean (true or false)
FTReset	Setting this parameter resets the state of the file transfer process; in particular, it can be used to recover from a “too many files” error	Boolean (true or false)
GPSAltitudeFilter	Determines whether GPS readings in which the altitude is suspicious are ignored	Boolean (true or false)
IGain	Integral Gain for PID Navigation	16-bit signed integer
InvalidGPSCount	Count of invalid GPS samples (not settable)	8-bit unsigned integer
iPIB	iPIB Power/Enable	Boolean (true or false)
iPIBRESET	Reset iPIB	Boolean (true or false)
ISatLimit	Integral Saturation Limit for PID Navigation	16-bit unsigned integer
MAGDEC	Magnetic Declination	8-bit signed integer
ModemOffRequiredAddress	The payload board address which is monitored and which allows the Iridium modem to be powered off as long as the board address is accessible	16-bit unsigned integer
ModemOffRequiredPort	The payload port that is monitored and which allows the Iridium modem to be powered off as long as the payload is communicating	8-bit unsigned integer (1..3)
ModemPower	If set and other parameters are set appropriately, the Iridium modem is turned off	Boolean (true or false)
ModemPowerAtBoot	Determines if the Iridium modem is powered on at boot (ModemPower)	Boolean (true or false)
Payload1 PL1	Payload #1 Power	Boolean (true or false)
Payload2 PL2	Payload #2 Power	Boolean (true or false)
Payload3 PL3	Payload #3 Power	Boolean (true or false)
PEP	Power Extension Port Power	Boolean (true or false)
PGain	Proportional Gain for PID Navigation	16-bit signed integer
PIDNav	PID Navigation Mode (Line Following) Enable	Boolean (true or false)
SBDAlwaysOn	SBD is always powered	Boolean (true or false)
SBDDefaultOn	SBD defaults to powered on and in use	Boolean (true or false)
Sub	Sub Power	Boolean (true or false)
SubPayload SubPL	Sub Payload Power	Boolean (true or false)
Water	Water Speed Sensor Power	Boolean (true or false)

Parameter Name(s)	Description	Type
Weather	Weather Station Power	Boolean (true or false)
XBEECHANNEL	XBee modem channel number (12-23)	8-bit unsigned integer
XBEEPANID	XBee modem PAN ID	16-bit unsigned integer

WCS address command

Sends a Write Console Script to another payload using the given address. The output of the command is the type of ACK or NAK followed by the text output. For example, the following sends a Write Console Script to address 0 (the Float C&C) and runs its “FCC –S” command, which lists the enumeration state:

```
$ ServerExec PayloadInterface 'WCS 0 fcc -s'
ACK3D:PL1:0n 0x0200(Full PIB v2.01.393) 0x2100(Sensor mgmt. compute v0.00.
001) 0x2000(Sensor mgmt. compute v2.01.000) PL2:0ff PL3:0ff iPIB:0n 0x1000
(v1.03.350) No UnTwist
```

The following sends a Write Console Script to address 0x200 (by convention, a PIB) and runs its “PIB –C” command, which lists its configuration:

```
$ ServerExec PayloadInterface 'WCS 0x200 pib -c'
ACK3D:PM1=COMM PM2=SMC PM3=EMPTY Power Down Delay = 1 seconds
```

PayloadInterface implements the following commands only when running the SV3 REST API:

HOSTGPS on-or-off

PayloadInterface typically does not poll the VMC for GPS unless the system GPS (typically published by GPSTListener) is off-line. This command can require PayloadInterface to poll the VMC for GPS by setting HOSTGPS mode on.

HOSTINVOKE {--board-address=source-address} script-command

“HOSTINVOKE” sends a script command to the VMC and returns the result. If a board address is specified, the command is executed as though it came from the specified address; otherwise, the command will appear to be from PayloadInterface. Note that when specifying a script command from WGMS, it must be prefixed with a percent sign (“%”); but this is not the case when using “HOSTINVOKE.”

KILLPOWER

“KILLPOWER” sends a message to the VMC to shut down power to the port. When this happens, PayloadInterface’s polling commands to the VMC will return indicating that the port is about to be powered off. PayloadInterface then requests the VMC to delay shutdown according to the “powerOffDelay” value in the XML file. PayloadInterface then sends out the Shutdown CORBA event and performs an orderly system shutdown -- using shutdown(8).

SETAMPSSW {--board-address=source-address} {switch-name} on-or-off

“SETAMPSSW” sends a message to the VMC requesting a power switch resident in the payload box to be turned on or off. One or more power switches can be associated with each payload in the JSON using the “hardware” and “connector” keys:

```
{ "boardAddress": 0,
  "corbaName": "PayloadInterface",
  "description": "Payload interface",
  "hardware": [ { "connector": "ICPwr1" } ],
```

```
"hostID":"1022102 SOMDM3730-20-2780AGCR-B 0213A00074",  
"interface":[{"swver":"2.1.0"}],  
"reportingAddress":8192,
```

In this case, the switch name is optional and command affects staall listed power switches. Otherwise, it affects only the indicated switch. Switches are named like "ICPwrn", where *n* ranges from 1 to the count of switches available.

Specifying a board address clarifies the payload executing the command (and the default switches associated with its registration.) If no board address is specified, the command is assumed to have come from PayloadInterface.

PPPHost Configuration

PPPHost establishes a Point-To-Point Protocol-based internet connection via an Iridium modem periodically or upon command.

Common XML file keys to edit

- **scpExecutorSettings** → **callInInterval** contains the number of seconds between automated call-ins (default 3600, or 1 hour) If this value is set to zero (0), connections are not made on a timed interval. However, the CallNow script (or executing "ServerExec PPPHost CALLNOW") causes a connection attempt to be made immediately.
- **scpExecutorSettings** → **scpVerb** contains the shell script command to execute just after the connection to the Internet is made. It is initially set to `./runscp 0002`, which causes a new sub-directory (`~/roots/200_PPPHost/workspace`) to be created and a "runme" script file to be copied from an LRI file server under the 0002 directory to the new workspace directory. The follow-up command is configured with the **runVerb** key.
- **scpExecutorSettings** → **runVerb** contains the shell script command to execute after successful execution of the **scpVerb** command. By default, it changes to the workspace directory and runs the "runme" file that was just copied.

Using this scheme, a shore-side server can input commands to be executed on call-in and vary the commands over time.

Configuring the RUDICS Modem

1. With the modem powered on and connected to an available serial port, establish a terminal session to the modem using HyperTerminal or putty (Windows) or GkTerm, minicom or screen (Linux) The default configuration should be able to operate at 19,200 baud, 8 data bits, 1 stop bit and no parity.
2. Reset the modem to factory default configuration by typing `AT&F0` (enter) and verify that the modem answers with `OK` or `0`
3. Set the baud rate to a fixed 19,200 baud by typing `AT+IPR=6,0` (enter) and verify that the modem answers with `OK` or `0`
4. Set 'verbose' mode by typing `AT&V1` (enter) and verify that the modem answers with `OK`
5. Set echo off by typing `AT&E0` (enter) and verify that the modem answers with `OK`

6. Save the configuration first typing AT&W0 (enter) and then AT&W1 (enter), verifying that the modem answer with OK after each command.
7. Cycle power to the modem and then type AT+IPR? (enter), verifying that it is still configured as set above (6,0)
8. Read the IMEI number from the modem by typing AT+CGSN (enter); if desired, record the resulting number for reference and compare with billing or provisioning records.
9. Read the ICC ID (Integrated Circuit Card Identification) or SIM card number by typing AT+CICCID (enter); the response should be a 19 or 20 digit number. Various errors may be returned if the SIM card is not properly installed: a SIM card is required for RUDICS. If desired, record the SIM card number for reference and compare with billing or provisioning records.

Configure the dialup script

The PPP daemon uses a dialer script to interact with the modem and establish the phone call. I used the chat program and the following script (/home/smc/roots/200_PPPhost/chat-iridium) to establish connection:

```
ABORT    'BUSY'
ABORT    'NO CARRIER'
ABORT    'ERROR'
ABORT    'NO DIALTONE'
ABORT    'NO ANSWER'
''       AT
OK       AT+CGSN
OK       AT+CICCID
OK       AT+CSQF
OK       AT+CREG?
OK       AT+CPIN?
OK       AT+CBST=71,0,1
OK       ATDT00881699999999
CONNECT  \d\c
```

Correct the telephone number to match the RUDICS gateway number provided by the Iridium vendor for use with this modem.

Configure the PPP connection script

The PPP connection script (/home/smc/200_PPPhost/runppp) uses the dialer script (above) to contact the Iridium gateway and then implements the Point-To-Point Protocol over the established connection to connect to the Internet. It contains the serial port name to be used when establishing the connection (typically /dev/ttyLX0) Edit this script to change the port name if necessary.

Manual connection to Iridium network

The following script is normally executed by PPPhost during connection, but it can be executed manually to set the system up via a terminal program. Note that the modem must be connected to an antenna with an unobstructed view of the sky.

1. With the modem connected the SMC (assuming port /dev/ttyLX0 and electrical switch 1), turn on power to the modem:

```
smc@omap35x_torpedo:~$ sudo SetPowerSwitch 1 1
SetPowerSwitch: 0x11
```

2. Enable RTS/CTS flow control by typing AT&K3 and verify that the modem answers with OK
3. Configure the modem to hang up on DTR transition from ON to OFF by typing AT&D2 (enter) and verify that the modem answers with OK
4. Prevent the modem from answering incoming calls by AT+SD=0 (enter) and verify that the modem answers with OK
5. Disable the SBD ring indicator by typing AT+SBDMTA=0 (enter) and verify that the modem answers with OK
6. Stop automated signal strength reports by typing AT+CIER=0,0,0 (enter) and verify that the modem answers with OK
7. Enable automated network registration reports by typing AT+CREG=2 (enter) and verify that the modem answers with OK
8. Get an immediate registration report by typing AT+CREG? and verify that the modem answers with a string formatted like +CREG:002,001,"891","ffff" followed by OK

The status number, which is the second number in the response, '001' in the example above should be interpreted as follows:

- 0 (not registered, not searching for new operator) or 3 (registration denied) indicate failure
- 1 (registered, home network) or 5 (registered, roaming) indicate success
- 2 (not registered, searching for new operator) or 4 (unknown/initial state) are normal waiting values: continue to monitor automated reports. Note that the Iridium network has some gaps that can delay registration for 5-10 minutes.

In case of failure, verify antenna connection or check modem provisioning with Iridium.

9. Disable automated registration reports by typing AT+CREG=0 (enter) and verify that the modem answers with OK
10. Get the status of the SIM card by typing AT+CPIN? (enter) and verify that the modem responds with +CPIN:SIM PIN2 message followed by OK. Other responses indicate that the SIM card is not usable and troubleshooting is required.
11. From the Linux command prompt, run the PPP connection script (./runppp &) and wait for the connection to be made.

```
smc@omap35x_torpedo:~/roots/200_PPPhost$ ./runppp &
[1] 361
smc@omap35x_torpedo:~/roots/200_PPPhost$ abort on (BUSY)
abort on (NO CARRIER)
abort on (ERROR)
abort on (NO DIALTONE)
abort on (NO ANSWER)
send (AT^M)
expect (OK)
^M
OK
-- got it

send (AT+CGSN^M)
```

```
expect (OK)
^M
^M
300025019999999^M
^M
OK
-- got it

send (AT+CICCID^M)
expect (OK)
^M
^M
89889999999999999^M
^M
OK
-- got it

send (AT+CSQF^M)
expect (OK)
^M
^M
+CSQF:5^M
^M
OK
-- got it

send (AT+CREG?^M)
expect (OK)
^M
^M
+CREG:000,001^M
^M
OK
-- got it

send (AT+CPIN?^M)
expect (OK)
^M
^M
+CPIN:SIM PIN2^M
^M
OK
-- got it

send (AT+CBST=71,0,1^M)
expect (OK)
^M
^M
OK
-- got it

send (ATDT0088169999999^M)
expect (CONNECT)
^M
^M
CONNECT
-- got it

send (\d)
Serial connection established.
using channel 1
Using interface ppp0
Connect: ppp0 <--> /dev/ttyLx0
.
.
.
... A large number of lines omitted ...
.
.
.
local IP address 192.168.13.51
```

remote IP address 192.168.13.254

12. Test the connection by pinging a well-known internet server; for example, `sudo ping www.google.com`

```
smc@omap35x_torpedo:~/roots/200_PPPhost$ sudo ping www.google.com
PING www.l.google.com (74.125.153.104) 56(84) bytes of data.
64 bytes from ty-in-f104.1e100.net (74.125.153.104): icmp_seq=1 ttl=47 time=1543 ms
64 bytes from ty-in-f104.1e100.net (74.125.153.104): icmp_seq=2 ttl=47 time=1520 ms
64 bytes from ty-in-f104.1e100.net (74.125.153.104): icmp_seq=3 ttl=47 time=1522 ms
64 bytes from ty-in-f104.1e100.net (74.125.153.104): icmp_seq=4 ttl=47 time=1601 ms
^C
--- www.l.google.com ping statistics ---
5 packets transmitted, 4 received, 20% packet loss, time 888ms
rtt min/avg/max/mdev = 1520.877/1547.182/1601.591/32.714 ms, pipe 2
```

13. When you are done with the connection, get a process listing (`ps -efH`) and look for the `pppd` (PPP daemon); kill the process:

```
smc@omap35x_torpedo:~/roots/200_PPPhost$ ps -efH
UID          PID    PPID  C STIME TTY          TIME CMD
root           2        0  0 05:43 ?           00:00:00 [kthreadd]
.
.
... a large number of lines omitted ...
.
.
smc          317        1  0 05:43 ttyDefault 00:00:00 -sh
smc          361       317  0 05:46 ttyDefault 00:00:00 /bin/sh ./runppp
root         362       361  0 05:46 ttyS0      00:00:00 pppd /dev/ttyLX0 19200 crt
smc          374       317  0 05:47 ttyDefault 00:00:00 ps -efH
smc@omap35x_torpedo:~/roots/200_PPPhost$ sudo kill 362
Terminating on signal 15
smc@omap35x_torpedo:~/roots/200_PPPhost$ Connect time 1.0 minutes.
Sent 772 bytes, received 1024 bytes.
sent [LCP TermReq id=0x5 "User request"]
rcvd [LCP TermAck id=0x5]
Connection terminated.
```

14. Turn off power to the modem:

```
smc@omap35x_torpedo:~/roots/200_PPPhost$ sudo SetPowerSwitch 1 0
SetPowerSwitch: 0x00
```

Setting up a key file for scp/ssh (dbclient) connection without password

The SMC provides the option to transfer files using `scp` and execute scripts using an embedded systems flavor of `ssh` called `dbclient`. The following procedure for creating an identity key file can be used to avoid using passwords:

1. From the `/home/smc/200_PPPhost` directory, generate a public/private key file pair as follows:

```
(RSA key pair) dropbearkey -t rsa -f keyfile | grep ssh-rsa > keyfile.pub
(DSS key pair) dropbearkey -t dss -f keyfile | grep ssh-dss > keyfile.pub
```

By convention, the name of the vehicle is used for `keyfile`. For example, if my vehicle is called "stingray", the resulting private key file is called "stingray" and the resulting public key file is called "stingray.pub"

2. Make a connection to the Internet manually (see above)
3. Use `scp` to copy the public key file (e.g. "stingray.pub") to the target account's ".ssh" directory; for example:

```
scp stingray.pub stingrayuser@bigserver.com:~.ssh
```

The first time that the server is contacted, scp prompts with a warning message like "Host 'bigserver.com' is not in the trusted hosts file" and ask for permission to proceed. Verify that you want to proceed, which adds the entry into the known_hosts file. Subsequent script executions will not be asked.

Enter the password for the target account when prompted.

4. On the target server (e.g. bigserver.com) and account (stingrayuser), replace ~/.ssh/authorized_keys with the new public key file; or if multiple vehicles are calling in, concatenate the files into ~/.ssh/authorized_keys so that any of them can connect.

5. Verify ssh/dbclient access without a password:

```
dbclient -i stingray stingrayuser@bigserver.com
```

6. Verify scp access without a password:

```
ls > testfile.txt
scp -i stingray testfile.txt stingrayuser@bigserver.com:~
```

SeabirdGPCTD Configuration

SeabirdGPCTD implements an interface to a Seabird Glider Payload CTD, with or without an attached oxygen sensor. Its primary features are:

- A command interface that allows the user to start or stop sampling, request status, send low diagnostics commands and configure logging and startup actions.
- On-board log of samples and commands and responses.
- Parameterized sampling scenario that specifies how samples are concatenated into telemetry reports, sleep time between reports and input line flush times before starting to sample.

The first instance of the program starts up under **/home/smc/roots/200_SBGPC TD**

The static configuration file is **SeabirdGCPTD-config.xml**; its salient configuration settings are:

- (1) The CORBA server name and type. As with every service, the server name must be unique, and the name is used whenever executing command via ServerExec:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE boost_serialization>
<boost_serialization signature="serialization::archive" version="10">
<rootlevel class_id="0" tracking_level="0" version="1">
  <sensorProgramSettings class_id="1" tracking_level="0" version="1">
    <serverName>SBGPCTD</serverName>
    <serverKind></serverKind>
```

- (2) The board ID and task ID, which make up the board address used externally (from WGMS or the SV3 web page) to send commands to the process, and the address stamped onto telemetry samples. It is recommended to set the board ID to 255, which means that the board ID configured in PayloadManager is used. The readable name should match the server name.

```
<payloadManagerClientSettings class_id="11" tracking_level="0" version="0">
  <serviceName>PayloadManager</serviceName>
  <serviceKind></serviceKind>
```

```

<deviceAddedEventName>PayloadDeviceAdded</deviceAddedEventName>
<deviceChangedEventName>PayloadDeviceChanged</deviceChangedEventName>
<deviceRemovedEventName>PayloadDeviceRemoved</deviceRemovedEventName>
<commonEventType>PayloadRegistrationEvent</commonEventType>
<commonEventDomain>LiquidR-SMC</commonEventDomain>
<payloadDescription class_id="12" tracking_level="0" version="2">
  <boardID>255</boardID>
  <taskID>38</taskID>
  <description>Seabird GPCTD interface</description>
  <ior></ior>
  <readableName>SBGPCTD</readableName>
  <descriptiveJSON></descriptiveJSON>
</payloadDescription>
</payloadManagerClientSettings>

```

- (3) The sensor is powered on and off using the powerOnVerb and the powerOffVerb. Use **sudo SetPowerSwitch *switch-number new-state*** to affect the state of the local power outputs, or substitute other commands² to affect appropriate power switch.

```

<sensorSettings class_id="14" tracking_level="0" version="1">
  <powerOffVerb>sudo SetPowerSwitch 3 0</powerOffVerb>
  <powerOnVerb>sudo SetPowerSwitch 3 1</powerOnVerb>
  <serialListenerSettings class_id="15" tracking_level="0" version="0">
    <maxReadSize>1024</maxReadSize>
    <intercharacterGapTimeoutMs>50</intercharacterGapTimeoutMs>
  </serialListenerSettings>
  <serialPortSettings class_id="16" tracking_level="0" version="0">
    <portName>/dev/ttyLX6</portName>
    <baudRate>115200</baudRate>
    <parity>NONE</parity>
    <characterSize>8</characterSize>
    <stopBits>1</stopBits>
    <flowControl>NONE</flowControl>
    <rts>1</rts>
    <dtr>1</dtr>
    <breakTimeoutMultiplier>1</breakTimeoutMultiplier>
  </serialPortSettings>
</sensorSettings>

```

The serial port device name (portName) and the baud rate to be used (baudRate) can also be set.

- (4) There is a setting for the logger that should be modified if there are additional instances. Specifically, the directory where the logs are stored should be changed to prevent output from different sensor instances from getting intermixed or overwritten:

```

<fileSetSettings class_id="14" tracking_level="0" version="0">
  <archiveCompressionType>BZIP2</archiveCompressionType>
  <archiveDirectory></archiveDirectory>
  <extension>.CSV</extension>
  <keepWritesTogether>1</keepWritesTogether>
  <maxBytesQueued>10000</maxBytesQueued>
  <maxFileSize>1000000</maxFileSize>
  <maxFiles>1000</maxFiles>
  <maxRetainTime>00:00:00</maxRetainTime>
  <prefix>SBGPCTD</prefix>
  <purgeStrategy>OLDEST</purgeStrategy>
  <workingDirectory>/mnt/XDATA/SBGPCTD</workingDirectory>
</fileSetSettings>

```

² For example, PayloadInterface implements some commands to affect AMPS power switches for SV3-style payloads (**SETAMPSSW**) or to set power switches using runtime parameters from SV2-style payloads (**SETPARAM.**) These can be executed using ServerExec.

The persistent configuration file, which holds values that can be changed through Write Console Script or ServerExec commands, is **SeabirdGCPTD-persist-config.xml**:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE boost_serialization>
<boost_serialization signature="serialization::archive" version="10">
<rootlevel class_id="0" tracking_level="0" version="1">
  <autoRunStr>false</autoRunStr>
  <dutyCycleSec>0</dutyCycleSec>
  <flushTimeSec>5</flushTimeSec>
  <loggingEnabledStr>true</loggingEnabledStr>
  <rawLoggingEnabledStr>false</rawLoggingEnabledStr>
  <reportBlockSize>8</reportBlockSize>
  <samplePeriodSec>60</samplePeriodSec>
</rootlevel>
</boost_serialization>
```

Note that the settings in the persistent configuration XML file are tracked with an MD5 signature file to detect corruption, **so this file cannot be edited by hand**. In the case where the MD5 signature doesn't match, all settings return to default values (as above.)

SeabirdGPCTD Telemetry Output

Two types of telemetry are output under payload type 0x70:

1. Telemetry sub-type 8 as CORBA message name "SBGPCTDSample" for samples with no O2 sensor installed
2. Telemetry sub-type 9 as CORBA message name "SBGPCTDSampleWithDO" for samples with O2 sensor installed.

The program is operating the sensor with real-world units, but types 8 and 9 assume that the sensor is reporting in engineering units. As a result, the equivalent engineering units are computed and both sets of values are included in the CORBA event:

Tag	Description	Type
SBGPCTDSample_ConductivityStr	The conductivity reading from the sensor exactly as output (S/m)	STRING
SBGPCTDSample_RawConductivity	Conductivity (Engineering units, use this for telemetry)	INT32
SBGPCTDSample_Conductivity	Conductivity string converted to float	SINGLE_FLOAT
SBGPCTDSample_TemperatureStr	The temperature reading from the sensor exactly as output (C)	STRING
SBGPCTDSample_RawTemperature	Temperature (Engineering units, use this for telemetry)	INT32
SBGPCTDSample_Temperature	Temperature string converted to float	SINGLE_FLOAT
SBGPCTDSample_RawPressure	Pressure (Engineering units, use this for telemetry)	INT32
SBGPCTDSample_PressureStr	The pressure reading from the sensor exactly as output (decibars)	STRING
SBGPCTDSample_Pressure	Pressure string converted to float	SINGLE_FLOAT
SBGPCTDSample_DissolvedOxygenStr	The dissolved oxygen reading from the sensor exactly as output (Hz)	STRING
SBGPCTDSample_RawDissolvedOxygen	Dissolved oxygen (Engineering units, use this for telemetry)	INT32
SBGPCTDSample_DissolvedOxygen	Dissolved oxygen string converted to float	SINGLE_FLOAT
SBGPCTDSample_O2Installed	True if an O2 sensor is installed, false if it is not installed	BOOLEAN

The following is the XML configuration to insert into PayloadInterface-telemetry-config.xml to convert the sensor reading to binary. Note that the

number of samples per report will be modified by the runtime configuration of the sensor.

```

<Element>
  <first boardID="0xFF" taskID="0x26" />
  <second>
    <key boardID="0xFF" taskID="0x26" />
    <payloadType value="0x00000070" />
    <eventNameToFormatCode>
      <Element>
        <first value="SBGPCTDSample" />
        <second value="8" />
      </Element>
      <Element>
        <first value="SBGPCTDSampleWithD0" />
        <second value="9" />
      </Element>
    </eventNameToFormatCode>
    <reportParams>
      <Element>
        <first value="8" />
        <second maxSamples="1" targetAddress="2">
          <reportHeaderFormat>
            <fields>
              <Element libField="Header91" />
              <Element libField="Raw" eventField="Telem-
LengthToFollow" />
              <Element libField="Raw" eventField="Telem-PayloadType"
/>
              <Element libField="Raw" eventField="Telem-BoardID" />
              <Element libField="Raw" eventField="Telem-TaskID" />
              <Element libField="FC8BinaryNormal" />
              <Element libField="RawINT8" eventField="Telem-
NumSamples" />
            </fields>
          </reportHeaderFormat>
          <sampleHeaderFormat>
            <fields>
              <Element libField="Raw" eventField="Telem-Latitude" />
              <Element libField="Raw" eventField="Telem-Longitude" />
              <Element libField="Raw" eventField="Telem-TimeSec" />
              <Element libField="Raw"
eventField="SBGPCTDSample_RawPressure" />
              <Element libField="Raw"
eventField="SBGPCTDSample_RawTemperature" />
              <Element libField="Raw"
eventField="SBGPCTDSample_RawConductivity" />
            </fields>
          </sampleHeaderFormat>
        </second>
      </Element>
      <Element>
        <first value="9" />
        <second maxSamples="1" targetAddress="2">
          <reportHeaderFormat>
            <fields>
              <Element libField="Header91" />
              <Element libField="Raw" eventField="Telem-
LengthToFollow" />
              <Element libField="Raw" eventField="Telem-PayloadType"
/>
              <Element libField="Raw" eventField="Telem-BoardID" />
              <Element libField="Raw" eventField="Telem-TaskID" />
              <Element libField="FC8BinaryNormal" />
              <Element libField="RawINT8" eventField="Telem-
NumSamples" />
            </fields>
          </reportHeaderFormat>
          <sampleHeaderFormat>
            <fields>

```

```

        <Element libField="Raw" eventField="Telem-Latitude" />
        <Element libField="Raw" eventField="Telem-Longitude" />
        <Element libField="Raw" eventField="Telem-TimeSec" />
        <Element libField="Raw"
eventField="SBGPCTDSample_RawPressure" />
        <Element libField="Raw"
eventField="SBGPCTDSample_RawTemperature" />
        <Element libField="Raw"
eventField="SBGPCTDSample_RawConductivity" />
        <Element libField="Raw"
eventField="SBGPCTDSample_RawDissolvedOxygen" />
    </fields>
</sampleHeaderFormat>
</second>
</Element>
</reportParams>
</second>
</Element>
```

SeabirdGPCTD Commands

SeabirdGPCTD implements the following commands:

ctd --autorun <on-or-off>

Enables (on/true) or disabled (off/false) logging of all messages and responses from the sensor.

ctd --expert <command>

Send one of the supported device commands and get the output from it. The supported commands are:

- dc (display calibration coefficients)
- ds (display status)
- getcc (get calibration coefficients)
- getcd (get configuration data: XML)
- getec (get event counters: XML)
- gethd (get hardware configuration: XML)
- getsd (get status data: XML)
- sl (send last sample)
- slp (send last pressure sample)

ctd --power <on-or-off>

Enables (on/true) or disabled (off/false) power to the sensor. Note that power is automatically turned on when sampling is started, so there's no need to send an additional command to power it on.

ctd --start { <period(sec)> { <samples-per-block> { <flush-time(sec)> { <off-time(sec)> } } } }

Turns on power to the sensor and starts taking readings, returning telemetry for each sample. If included, the parameters overwrite the currently-configured values (the new values are saved persistently.)

- **period(sec)** is the number of seconds between samples. Note that this value cannot be over 14 seconds if an O2 sensor is installed (see GPCTD manual for further detail.)
- **samples-per-block** is the number of samples per telemetry sample, and number of samples between imposing an off time.
- **flush-time(sec)** is the number of seconds to run the pump (to get representative material in the line) before starting to sample.
- **off-time(sec)** is the number of seconds to suspend pumping and sampling between blocks. If this value is zero, sampling is continuous.

ctd --status

Returns a message representing the state of the sensor. For example:

```
$ ServerExec SBGPCTD "ctd --status"
OK Sampling Port=/dev/ttyLX6 Enable=true Pwr=true Intervl=1 sec Size=10 smpl/pkt
Off-Tm=5 sec Flush-Tm=3 Oxygen=true AutoRun=true Logging=samples
```

ctd --stop

Stops the sampling loop, but leaves the sensor powered (in low power state.)

SimpleSBDQueue Configuration

SimpleSBDQueue is a service that interfaces an Iridium 9522B or 9523 SBD (Short Burst Data) satellite modem and connects it to the SMC's CORBA infrastructure. Messages sent to the service are stored in a queue and are sent sequentially as SBD messages as fast as the policy allows; messages received over SBD are sent as CORBA events.

Commands

SimpleSBDQueue supports the following commands:

SSQ --run <filename>

Opens *filename* and runs any commands it contains.

SSQ --enqueue [--priority= <priority>] [--senderid= <senderid>] [--sendertransaction= <sendertransaction>] [--queue= <position>] [--ttl= <timetolive>] <message>

Queues up a message to be sent out over the satellite network. The content of the SBD buffer is provided by *message*, which is a hexadecimal string representing the bytes. Other variables that may be provided are:

- *priority* (unsigned 16-bit number), which represents how important the message is; messages with lower priority numbers are sent sooner. The default value is **3**.

- *senderid* (unsigned 16-bit number), which represents the board address of the sending process. Status outputs related to this message include the *senderid*. The default value is **0xFFFF**.
- *sendertransaction* (unsigned 32-bit number), which is meant to allow the sender to distinguish between sent messages. Status outputs related to this message include the *sendertransaction*. The default value is **0xFFFFFFFF**.
- *position* is where to place the message in the queue. Valid values are:
 - o end: store the message at the end of the list
 - o front: store the message so that it will be transmitted next
 - o latest: the message will be sent next unless another message comes in, in which case it goes to the end of the queue
 The default value for *position* is "end"
- *timetolive* (unsigned 32-bit number) is how long to let the message sit in the queue before timing it out (milliseconds, 0=no timeout.) The default value is **0**.

SSQ --mbcheck

Triggers a mailbox check, which causes the modem to be activated and an empty transmission to be sent. This has the effect of checking the status of the "mailbox" and reading the next message if any are pending. This is not typically needed, but can be useful for debugging.

SSQ --start

Starts the background execution of the protocol, which periodically checks for messages from the satellite network and sends any pending messages.

SSQ --stop

Stops the background protocol. No attempt will be made to check for messages from the satellite network and no messages will be sent.

CORBA Events

SimpleSBDQueue reports the following CORBA events:

EventName	TypeName	DomainName	Class
SBDTransmitDisposition	SBDTransmitDisposition	LiquidR	SBDPacketEvent
This event is sent after each transmission to indicate the MOMSN and status of the transmission, including the original sender ID and transaction for reference.			
SBDStatsUpdate	SBDChannelStats	LiquidR	SBDStatisticsEvent
This event is sent after each connection attempt to indicate the status, success rate and reliability of the connection.			
SBDReceivedPacket	SBDPacketEvent	LiquidR	SBDPacketEvent
This event is sent whenever a packet is received from the SBD modem.			

SimpleSBDQueue can receive the following CORBA events:

EventName	TypeName	DomainName	Class
SBDEnqueuePacket	SBDPacketEvent	LiquidR	SBDPacketEvent
This event can be sent to SimpleSBDQueue as alternative way to queue up a message for transmission.			

XML Configuration

The configuration file (SimpleSBDQueue-config.xml), which must be located in the startup directory (typically /home/smc/roots/200_SimpleSBDQueue), conforms to the format of other configuration files, but here are a few items of interest:

```
<autoStart>true</autoStart>
```

Configures whether to automatically begin operating on startup.

```
<concatenateSentPackets>false</concatenateSentPackets>
```

Configures whether to concatenate messages to maximize use of the buffer before transmitting.

```
<minOffTimeMS>3000</minOffTimeMS>
<maxOnTimeMS>300000</maxOnTimeMS>
```

Configures the minimum amount of time that the modem must remain off when it is turned off (*minOffTimeMS*, milliseconds) and the maximum amount of time that it can remain on (*maxOnTimeMS*, milliseconds.)

```
<dutyCycleModem>true</dutyCycleModem>
```

Configures whether modem power should be cycled on and off to reduce power usage (true) or whether it should remain on all the time (false.)

```
<networkAvailableTimeoutMS>60000</networkAvailableTimeoutMS>
<networkFailDelayMS>120000</networkFailDelayMS>
```

networkAvailableTimeoutMS configures how long (milliseconds) to wait for the satellite network to be available before declaring a network failure. If a network failure is declared, *networkFailDelayMS* configures how long (milliseconds) to wait before attempting communication again.

```
<mailboxCheckIntervalMS>60000</mailboxCheckIntervalMS>
```

Configures how frequently (milliseconds of idle delay) to check for messages from the satellite network. Note that if messages are being sent to the satellite network more frequently, mailbox checks will not be performed.

```
<commFailureDelayMS>120000</commFailureDelayMS>
```

If a communication failure with the modem is declared, *commFailureDelayMS* configures how long to wait (milliseconds) before attempting further communication.

```
<useRingIndicator>false</useRingIndicator>
```

Determines whether ring indication should be used to indicate when a message is waiting. Note that this is only effective if the modem is left turned on. **Note: waiting for ring indicator has not been implemented yet.**

```
<networkTimeoutSeconds>1200</networkTimeoutSeconds>
```

If the modem cannot contact the satellite network for *networkTimeoutSeconds* or more, modem communication is declared to be unreliable (to client applications.)

timelimit2

The standard **timelimit** utility limits the clock runtime of a process to a given number of seconds; for example:

```
exec timelimit -pq -s 9 -t 600 scp $IDENT -P 10501
vehicle@12.156.66.48:$DIRPART/runme workspace/runme
```

In this case, the “scp” command is limited to 600 seconds, and then is killed if it is not complete yet. Some issues with this were found, though, when the command creates a child task. In this configuration, **timelimit** only kills the top level process: the children continue to run.

To resolve this, **timelimit2** uses some classes already in use in the SMC framework classes to recursively kill the children as well as the parent. The syntax is as follows:

```
timelimit2 -timeout=ttt {--signal=sss} {--debug} ...command-line...
```

The timeout value (*ttt*) is the maximum number of seconds to allow the process to run; the optional signal value (*sss*) is the signal number to use when killing processes, which defaults to 9 (SIGKILL)

VirtualRelay Configuration

The VirtualRelay application is a module that uses the Virtual Relay protocol to communicate with WGMS over a TCP/IP connection. Samples methods of transports include WiFi, Ethernet, or a cell modem. These communication transports are managed by ConnectionManager.

VirtualRelay is dependent on ConnectionManager. VirtualRelay is also dependent on the FCC firmware, which will be in a future release (version 2.0.973 or newer).

The VirtualRelay application is started from

```
~ /roots/200_VirtualRelay /
```

And is configured by the XML file:

```
/roots/200_VirtualRelay/VirtualRelay-config.xml
```

VirtualRelay is assigned the board ID 32 (0x20) and the task ID 170 (0xAA). Each channel is assigned its own task ID as well. For example, if a cell modem is configured, it should take task ID 171 (0xAB).

Common XML file keys to edit

A typical configuration for the controller settings is as follows:

```
<controllerSettings>
  <count>1</count>
  <item_version>0</item_version>
  <item>
    <host>test.liquidr.com</host>
    <hostPort>643</hostPort>
    <commsTaskID>171</commsTaskID>
    <sleepTime>180</sleepTime>
    <retries>10</retries>
    <retryRate>20</retryRate>
    <timeoutSeconds>5</timeoutSeconds>
    <messageManagerSettings>
      <telemetryInterrupt>1</telemetryInterrupt>
      <ackInterrupt>1</ackInterrupt>
      <telemetryTimeoutCount>4</telemetryTimeoutCount>
      <protocolRevision>32</protocolRevision>
    </messageManagerSettings>
  </item>
</controllerSettings>
```

```
test32.xml</backupFilename>
      <backupFilename>savedMessages-
    </messageManagerSettings>
  </item>
</controllerSettings>
```

- **count**: This is the number of channels that the Virtual Relay will use.
- **host**: This is where the Virtual Relay will try to connect to send and receive messages. Typical values are test.liquidr.com (the test server) or packets.wgms.com (the production server).
- **hostPort**: The port that Virtual Relay uses to connect to the host. This is 643 for the test server and 443 for the production server.
- **commsTaskID**: This is the task ID that this particular channel uses to communicate with the FCC. Note that this should be different than the Virtual Relay's task ID of 170 (0xAA) and unique for every channel.
- **sleepTime**: This is the amount of time, in seconds, that the Virtual Relay will wait after the last communication with the host before starting to attempt a new session. Use this parameter to control how the channel is duty-cycled. A setting of 0 will disable duty cycling.
- **retries**: This is the number of times the Virtual Relay will try to connect to the host before giving up and waiting the amount of time specified by *sleepTime*.
- **retryRate**: This is the amount of time, in seconds, that the Virtual Relay will wait between attempted connections to the host. Make sure to set *retries* and *retryRate* such that the channel being used has enough time to be initialized (for example, if a cell modem takes 60 seconds to start up, make sure *retries***retryRate* is greater than 60).
- **timeoutSeconds**: This is the timeout for the socket connection made between the Virtual Relay and the host. This should be set to five seconds at a minimum, but it can be set higher if there is latency in the connection (eg, a non-wired connection).
- **telemetryInterrupt**: This specifies whether or not a telemetry message will interrupt the channel if it is in sleep mode. (0=off, 1=on).
- **ackInterrupt**: This specifies whether or not an ack will interrupt the channel if it is in sleep mode. (0=off, 1=on).
- **telemetryTimeoutCount**: If the Virtual Relay accumulates this many telemetry messages without being able to communicate to the host, it will tell the FCC to default back to Iridium.
- **protocolRevision**: This is the protocolRevision that this channel uses to communicate. This determines which channel ConnectionManager is asked to turn on/off and the channel that this Virtual Relay reports as to the host.

- **backupFilename:** In the event that the SMC is powered off while there are messages waiting to be sent back to the host, they will be stored in this file and read when the Virtual Relay next starts up.

Typical Virtual Relay configurations

Setting	WiFi/Ethernet		Cell Modem	
	Production	Test	Production	Test
Host	packets.wgms.com	test.liquidr.com	packets.wgms.com	test.liquidr.com
hostPort	443	643	443	643
sleepTime	180	180	200	200
retries	10	10	15	15
retryRate	20	20	20	20
timeoutSeconds	5	5	15	15
protocolRevision	32	32	33	33

Commands

A user can change some of the Virtual Relay settings through commands sent to the process VirtualRelay via ServerExec, or to the Virtual Relay task (task ID 170, ox 0xAA). These settings will be saved in the xml configuration file and will persist after a reboot.

- **setsleeptime *channelTaskID newSleepTime***
Changes the sleepTime configuration for the channel specified by the task ID. For example, a sample command may be:

```
setsleeptime 171 60
```

- **setretryrate *channelTaskID newChannelTaskID***
Changes the retryRate configuration for the channel specified by the task ID. For example, a sample command may be:

```
setretryrate 171 15
```

- **setnumretries *channelTaskID newNumRetries***
Changes the retries configuration for the channel specified by the task ID. For example, a sample command may be:

```
setnumretries 171 5
```

- **setteleminterrupt *channelTaskID newTelemInterrupt***
Changes the telemetryInterrupt configuration for the channel specified by the task ID. For example, a sample command may be:

```
setteleminterrupt 171 0
```

- `setackinterrupt channelTaskID newAckInterrupt`
Changes the `ackInterrupt` configuration for the channel specified by the task ID. For example, a sample command may be:

```
setackinterrupt 171 0
```

- `settimeout channelTaskID newTimeout`
Changes the `timeoutSeconds` configuration for the channel specified by the task ID. For example, a sample command may be:

```
settimeout 171 10
```

VemcoVR2C Configuration

The VemcoVR2C application is a module capable of interfacing to, initializing, and receiving data, over a serial port, from a Vemco VR2C sensor. All communications and data received from the sensor is logged in the application's log file `~/log/VemcoVR2C.log` and all Tag detections reported by the sensor (read from the serial port as ASCII strings) are also sent as telemetry events **VR2CTagReport**. The VemcoVR2C application is started from:

```
~/roots/200_VemcoVR2C/
```

And is configured by the XML file:

```
~/roots/200_VemcoVR2C/VemcoVR2C-config.xml
```

The VemcoVR2C instance is configured with the `serverName` **VR2C**, and assigned the `boardId` 32 (hexadecimal 20) and `taskId` 21 (hexadecimal 15).

Common XML file keys to edit

- **serialPortSettings** for VemcoVR2C, the default serial port is `/dev/ttyLX4`. By default it is set to communicate with a baudrate of 9600, parity NONE, 1 stopbit, characterSize 8, and flowControl NONE. Ensure that the serial connector cables are plugged into the specified serial port on the SMC.
- **powerOnVerb** and **powerOffVerb** for VemcoVR2C, the command string references power switch. Ensure that the power cable is plugged into the specified power switch on the SMC.
- **VR2CserialNumber** this is used to identify responses from the sensor, and is specified as a six digit number. For example **450006**
- **VR2CcmdPrefix** this string is used to construct the command to be sent to the sensor and includes special characters, the serial number and a checksum, as per a fixed format. For example: ***450006.0#15,**
- **VR2CwakeupRetries** number of times to re-try the wakeup sequence on the sensor if no response is received. Default setting is **1**.
- **VR2CwakeupDelay** time, in milliseconds, after which a response to the wakeup command should be received. Default setting is **150**.
- **VR2CresponseDelay** time, in milliseconds, after which a response to any command (except wakeup) should be received. Default setting is **2000**.

- **TagPrefix** an ASCII string depicting the start of a valid tag detection field in the message received from the sensor. Default setting is **A69-**.
- **ReportChokeMethod** a number specifying what method, if any, to use for regulating the frequency of reporting tag detection samples as telemetry events. A value of 0 indicates no regulation. A value of 1 indicates a minimum gap (delay) between successive events as defined below. All tag detection samples will be reported, but with the minimum delay between successive reports, if the samples arrive more frequently. Default setting is **1**.
- **ReportDelaySeconds** time, in seconds, for the gap/delay between successive telemetry events of tag detection samples if ReportChokeMethod 1 is selected. Default setting is **60**.
- **ReportDelaySeconds** time, in seconds, for the gap/delay between successive telemetry events of tag detection samples if ReportChokeMethod 1 is selected. Default setting is **60**.

Using the VemcoVR2C

The VemcoVR2C application, for this demo implementation, performs a power On of the sensor, initializes it as per a set study sequence for real time monitoring of tag detections, and then monitors all serial data received from the sensor for tag detections which are packaged and sent as telemetry events.

WetLabsSensorSimulator

The utility `~/bin/WetLabsSensorSimulator` is designed to generate simulated Eco Puck samples, provide configuration and menu responses to sensor commands, and emulate the sensor's start and stop states when responding to these commands. The commands are received and simulated responses sent over a pseudo serial port, specified when starting the utility.

In order use the simulated communications, the WetLabsSensor application must be configured for communicating with an Eco Puck sensor (configured in `~/roots/200_EcoSensor`) and using the pseudo serial port specified when invoking the WetLabsSensorSimulator.

Example:

```
WetLabsSensorSimulator /home/smc/roots/200_EcoSensor/pseudo
```

Appendix 1 - Sample Application Configuration File

The following is a sample application configuration file listing taken from the GPS Listener process (GPSListener-config.xml):

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE boost_serialization>
<boost_serialization signature="serialization::archive" version="10">
  <rootlevel class_id="0" tracking_level="0" version="7">
    <loggerSettings class_id="1" tracking_level="0" version="0">
      <logLevel>LT_WARNING</logLevel>
      <processName>GPSListener</processName>
      <isFileDes>1</isFileDes>
      <fileDes>1</fileDes>
      <fileName/>
      <pipeName/>
    </loggerSettings>
    <serialPortSettings class_id="2" tracking_level="0" version="0">
      <portName>/dev/tty01</portName>
      <baudRate>4800</baudRate>
      <parity>NONE</parity>
      <characterSize>8</characterSize>
      <stopBits>1</stopBits>
      <flowControl>NONE</flowControl>
      <rts>1</rts>
      <dtr>1</dtr>
      <breakTimeoutMultiplier>1</breakTimeoutMultiplier>
    </serialPortSettings>
    <maxString>1000</maxString>
    <readTimeoutMs>50</readTimeoutMs>
    <serverName>GPSListener</serverName>
    <serverKind/>
    <dateSetDirectly>1</dateSetDirectly>
    <dateVerb>sudo date --utc {mm}{dd}{HH}{MM}{yyyy}.{SS}</dateVerb>
    <hwClockVerb>sudo hwclock -w</hwClockVerb>
    <defaultOutputFileName>/mnt/XDATA/GPS/sensor.out</defaultOutputFileName>
    <deltaToSetTimeMs>100</deltaToSetTimeMs>
    <deltaToSetHwClockMs>0</deltaToSetHwClockMs>
    <rmsAccuracyThreshold>50</rmsAccuracyThreshold>
    <gpsEventSpecs class_id="3" tracking_level="0" version="0">
      <count>2</count>
      <item_version>1</item_version>
      <item class_id="4" tracking_level="0" version="1">
        <divisor>5</divisor>
        <divisorMode>0</divisorMode>
        <eventSettings class_id="5" tracking_level="0" version="0">
          <eventName>GPSSample</eventName>
          <eventType class_id="6" tracking_level="0" version="0">
            <typeName>GPSSample</typeName>
            <domainName>LiquidR-SMC</domainName>
          </eventType>
        </eventSettings>
      </item>
      <item>
        <divisor>1</divisor>
        <divisorMode>0</divisorMode>
        <eventSettings>
          <eventName>GPSSample1Sec</eventName>
          <eventType>
            <typeName>GPSSample</typeName>
            <domainName>LiquidR-SMC</domainName>
          </eventType>
        </eventSettings>
      </item>
    </gpsEventSpecs>
    <tryBaudRates>
      <count>2</count>
    </tryBaudRates>
  </rootlevel>
</boost_serialization>
```

```

    <item_version>0</item_version>
    <item>4800</item>
    <item>9600</item>
  </tryBaudRates>
  <setTimeOnPPS>true</setTimeOnPPS>
  <ppsGPIO>167</ppsGPIO>
  <systemGPSOffline>
    <eventName>SystemGPSOffline</eventName>
    <eventType>
      <typeName>EventBase</typeName>
      <domainName>LiquidR-SMC</domainName>
    </eventType>
  </systemGPSOffline>
  <systemGPSOnline>
    <eventName>SystemGPSOnline</eventName>
    <eventType>
      <typeName>EventBase</typeName>
      <domainName>LiquidR-SMC</domainName>
    </eventType>
  </systemGPSOnline>
  <hostGPSEventSettings>
    <eventName>HostGPSSample</eventName>
    <eventType>
      <typeName>GPSSample</typeName>
      <domainName>LiquidR-SMC</domainName>
    </eventType>
  </hostGPSEventSettings>
  <eventTransceiverSettings class_id="8" tracking_level="0" version="2">
    <eventServiceName>EventManager</eventServiceName>
    <eventServiceKind/>
    <receiveEnable>1</receiveEnable>
    <aliasToEvent class_id="9" tracking_level="0" version="0">
      <count>0</count>
      <item_version>0</item_version>
    </aliasToEvent>
    <eventFilterEnabled>1</eventFilterEnabled>
    <eventNameFilter class_id="10" tracking_level="0" version="0">
      <count>1</count>
      <item_version>0</item_version>
      <item>HostGPSSample</item>
    </eventNameFilter>
    <eventTypeFilter class_id="11" tracking_level="0" version="0">
      <count>0</count>
      <item_version>0</item_version>
    </eventTypeFilter>
    <displayReceivedEvents>0</displayReceivedEvents>
    <fullyDescribeReceivedEvents>0</fullyDescribeReceivedEvents>
    <doNotDisplayList>
      <count>0</count>
      <item_version>0</item_version>
    </doNotDisplayList>
  </eventTransceiverSettings>
  <pmClientSettings class_id="12" tracking_level="0" version="0">
    <serviceName>PayloadManager</serviceName>
    <serviceKind/>
    <deviceAddedEventName>PayloadDeviceAdded</deviceAddedEventName>
    <deviceChangedEventName>PayloadDeviceChanged</deviceChangedEventName>
    <deviceRemovedEventName>PayloadDeviceRemoved</deviceRemovedEventName>
    <commonEventType>PayloadRegistrationEvent</commonEventType>
    <commonEventDomain>LiquidR-SMC</commonEventDomain>
    <payloadDescription class_id="13" tracking_level="0" version="2">
      <boardID>255</boardID>
      <taskID>241</taskID>
      <description>GPS event source and time updater</description>
      <ior>(This will be overwritten)</ior>
      <readableName>GPSListener</readableName>
      <descriptiveJSON/>
    </payloadDescription>
  </pmClientSettings>
</rootlevel>

```


Appendix 2 - Sample Telemetry Configuration

The following listing is a sample configuration for the telemetry reported by PayloadInterface (PayloadInterface-telemetry-config.xml):

```
<?xml version="1.0"?>
<root>
  <publishers>
    <Element>
      <first boardID="0xFF" taskID="0xF0"/>
      <second>
        <key boardID="0xFF" taskID="0xF0"/>
        <payloadType value="0x0000007B"/>
        <eventNameToFormatCode>
          <Element>
            <first value="MYTELEMEVENT"/>
            <second value="0"/>
          </Element>
        </eventNameToFormatCode>
        <reportParams>
          <Element>
            <first value="0"/>
            <second maxSamples="2" targetAddress="2">
              <reportHeaderFormat>
                <fields>
                  <Element libField="Header91"/>
                  <Element libField="Raw" eventField="Telem-LengthToFollow"/>
                  <Element libField="Raw" eventField="Telem-PayloadType"/>
                  <Element libField="Raw" eventField="Telem-BoardID"/>
                  <Element libField="Raw" eventField="Telem-TaskID"/>
                  <Element libField="FC0BinaryNormal"/>
                  <Element libField="Raw" eventField="Telem-NumSamples"/>
                </fields>
              </reportHeaderFormat>
              <sampleHeaderFormat>
                <fields>
                  <Element libField="Raw" eventField="Telem-Latitude"/>
                  <Element libField="Raw" eventField="Telem-Longitude"/>
                  <Element libField="Raw" eventField="Telem-TimeSec"/>
                  <Element libField="DummySample"/>
                </fields>
              </sampleHeaderFormat>
            </second>
          </Element>
        </reportParams>
      </second>
    </Element>
    <Element>
      <first boardID="0xFF" taskID="0xF3"/>
      <second>
        <key boardID="0xFF" taskID="0xF3"/>
        <payloadType value="0x0000007D"/>
        <eventNameToFormatCode>
          <Element>
            <first value="BenthosModemRangeTelemetry"/>
            <second value="0"/>
          </Element>
        </eventNameToFormatCode>
        <reportParams>
          <Element>
            <first value="0"/>
            <second maxSamples="1" targetAddress="2">
              <reportHeaderFormat>
                <fields>
                  <Element libField="Header91"/>
                  <Element libField="Raw" eventField="Telem-LengthToFollow"/>
                  <Element libField="Raw" eventField="Telem-PayloadType"/>
                  <Element libField="Raw" eventField="Telem-BoardID"/>
                  <Element libField="Raw" eventField="Telem-TaskID"/>
                </fields>
              </reportHeaderFormat>
            </second>
          </Element>
        </reportParams>
      </second>
    </Element>
  </publishers>
</root>
```

```

        <Element libField="FC0BinaryNormal"/>
        <Element libField="RawINT8" eventField="Telem-NumSamples"/>
    </fields>
</reportHeaderFormat>
<sampleHeaderFormat>
    <fields>
        <Element libField="Raw" eventField="Telem-Latitude"/>
        <Element libField="Raw" eventField="Telem-Longitude"/>
        <Element libField="Raw" eventField="Telem-TimeSec"/>
        <Element libField="Raw" eventField="ATRSample"/>
        <Element libField="Raw" eventField="ATXSample0"/>
        <Element libField="Raw" eventField="ATXSample1"/>
        <Element libField="Raw" eventField="ATXSample2"/>
        <Element libField="Raw" eventField="ATXSample3"/>
        <Element libField="Raw" eventField="ATXSample4"/>
        <Element libField="Raw" eventField="ATXSample5"/>
    </fields>
</sampleHeaderFormat>
</second>
</Element>
</reportParams>
</second>
</Element>
<Element>
    <first boardID="0xFF" taskID="0x12"/>
    <second>
        <key boardID="0xFF" taskID="0x12"/>
        <payloadType value="0x0000007B"/>
        <eventNameToFormatCode>
            <Element>
                <first value="EcoPuckSample"/>
                <second value="0"/>
            </Element>
        </eventNameToFormatCode>
        <reportParams>
            <Element>
                <first value="0"/>
                <second maxSamples="2" targetAddress="2">
                    <reportHeaderFormat>
                        <fields>
                            <Element libField="Header91"/>
                            <Element libField="Raw" eventField="Telem-LengthToFollow"/>
                            <Element libField="Raw" eventField="Telem-PayloadType"/>
                            <Element libField="Raw" eventField="Telem-BoardID"/>
                            <Element libField="Raw" eventField="Telem-TaskID"/>
                            <Element libField="FC0BinaryNormal"/>
                            <Element libField="RawINT8" eventField="Telem-NumSamples"/>
                        </fields>
                    </reportHeaderFormat>
                    <sampleHeaderFormat>
                        <fields>
                            <Element libField="Raw" eventField="Telem-Latitude"/>
                            <Element libField="Raw" eventField="Telem-Longitude"/>
                            <Element libField="Raw" eventField="Telem-TimeSec"/>
                            <Element libField="Raw" eventField="col1"/>
                            <Element libField="Raw" eventField="col2"/>
                            <Element libField="Raw" eventField="col3"/>
                            <Element libField="Raw" eventField="col4"/>
                            <Element libField="Raw" eventField="col5"/>
                            <Element libField="Raw" eventField="col6"/>
                            <Element libField="Raw" eventField="col7"/>
                            <Element libField="Raw" eventField="col8"/>
                            <Element libField="Raw" eventField="col9"/>
                        </fields>
                    </sampleHeaderFormat>
                </second>
            </Element>
        </reportParams>
    </second>
</Element>
</Element>

```

```

<first boardID="0xFF" taskID="0x13"/>
<second>
  <key boardID="0xFF" taskID="0x13"/>
  <payloadType value="0x0000007C"/>
  <eventNameToFormatCode>
    <Element>
      <first value="CStarSample"/>
      <second value="0"/>
    </Element>
  </eventNameToFormatCode>
  <reportParams>
    <Element>
      <first value="0"/>
      <second maxSamples="2" targetAddress="2">
        <reportHeaderFormat>
          <fields>
            <Element libField="Header91"/>
            <Element libField="Raw" eventField="Telem-LengthToFollow"/>
            <Element libField="Raw" eventField="Telem-PayloadType"/>
            <Element libField="Raw" eventField="Telem-BoardID"/>
            <Element libField="Raw" eventField="Telem-TaskID"/>
            <Element libField="FC0BinaryNormal"/>
            <Element libField="RawINT8" eventField="Telem-NumSamples"/>
          </fields>
        </reportHeaderFormat>
        <sampleHeaderFormat>
          <fields>
            <Element libField="Raw" eventField="Telem-Latitude"/>
            <Element libField="Raw" eventField="Telem-Longitude"/>
            <Element libField="Raw" eventField="Telem-TimeSec"/>
            <Element libField="Raw" eventField="col1"/>
            <Element libField="Raw" eventField="col2"/>
            <Element libField="Raw" eventField="col3"/>
            <Element libField="Raw" eventField="col4"/>
            <Element libField="Raw" eventField="col5"/>
            <Element libField="Raw" eventField="col6"/>
          </fields>
        </sampleHeaderFormat>
      </second>
    </Element>
  </reportParams>
</second>
</Element>
<Element>
  <first boardID="0xFF" taskID="0x22"/>
  <second>
    <key boardID="0xFF" taskID="0x22"/>
    <payloadType value="0x0000007E"/>
    <eventNameToFormatCode>
      <Element>
        <first value="MOSESampleComplete"/>
        <second value="0"/>
      </Element>
    </eventNameToFormatCode>
    <reportParams>
      <Element>
        <first value="0"/>
        <second maxSamples="1" targetAddress="2">
          <reportHeaderFormat>
            <fields>
              <Element libField="Header91"/>
              <Element libField="Raw" eventField="Telem-LengthToFollow"/>
              <Element libField="Raw" eventField="Telem-PayloadType"/>
              <Element libField="Raw" eventField="Telem-BoardID"/>
              <Element libField="Raw" eventField="Telem-TaskID"/>
              <Element libField="FC0BinaryNormal"/>
              <Element libField="RawINT8" eventField="Telem-NumSamples"/>
            </fields>
          </reportHeaderFormat>
          <sampleHeaderFormat>
            <fields>

```

```
<Element libField="Raw" eventField="Telem-Latitude"/>
<Element libField="Raw" eventField="Telem-Longitude"/>
<Element libField="Raw" eventField="Telem-TimeSec"/>
<Element libField="Raw" eventField="MOSECompleteEvent_Hs"/>
<Element libField="Raw" eventField="MOSECompleteEvent_Tav"/>
<Element libField="Raw" eventField="MOSECompleteEvent_Tp"/>
<Element libField="Raw" eventField="MOSECompleteEvent_Dirp"/>
eventField="MOSECompleteEvent_NumSpectra"/>
<Element libField="Raw"
eventField="MOSECompleteEvent_SamplesPerSpectrum"/>
<Element libField="Raw"
eventField="MOSECompleteEvent_SampleGaps"/>
</fields>
</sampleHeaderFormat>
</second>
</Element>
</reportParams>
</second>
</Element>
<Element>
<first boardID="0xFF" taskID="0x15"/>
<second>
<key boardID="0xFF" taskID="0x15"/>
<payloadType value="0x0000007F"/>
<eventNameToFormatCode>
<Element>
<first value="VR2CTagReport"/>
<second value="0"/>
</Element>
</eventNameToFormatCode>
<reportParams>
<Element>
<first value="0"/>
<second maxSamples="1" targetAddress="2">
<reportHeaderFormat>
<fields>
<Element libField="Header91"/>
<Element libField="Raw" eventField="Telem-LengthToFollow"/>
<Element libField="Raw" eventField="Telem-PayloadType"/>
<Element libField="Raw" eventField="Telem-BoardID"/>
<Element libField="Raw" eventField="Telem-TaskID"/>
<Element libField="FC0ASCIINormal"/>
<Element libField="RawINT8" eventField="Telem-NumSamples"/>
</fields>
</reportHeaderFormat>
<sampleHeaderFormat>
<fields>
<Element libField="Raw" eventField="Telem-Latitude"/>
<Element libField="Raw" eventField="Telem-Longitude"/>
<Element libField="Raw" eventField="Telem-TimeSec"/>
<Element libField="Raw" eventField="StringEvent-String"/>
</fields>
</sampleHeaderFormat>
</second>
</Element>
</reportParams>
</second>
</Element>
<Element>
<first boardID="0xFF" taskID="0x17"/>
<second>
<key boardID="0xFF" taskID="0x17"/>
<payloadType value="0x00000060"/>
<eventNameToFormatCode>
<Element>
<first value="C3SampleHeader"/>
<second value="0"/>
</Element>
<Element>
<first value="C3SampleData"/>
```

```

        <second value="1"/>
    </Element>
</eventNameToFormatCode>
<reportParams>
    <Element>
        <first value="0"/>
        <second maxSamples="1" targetAddress="2">
            <reportHeaderFormat>
                <fields>
                    <Element libField="Header91"/>
                    <Element libField="Raw" eventField="Telem-LengthToFollow"/>
                    <Element libField="Raw" eventField="Telem-PayloadType"/>
                    <Element libField="Raw" eventField="Telem-BoardID"/>
                    <Element libField="Raw" eventField="Telem-TaskID"/>
                    <Element libField="FC0ASCIINormal"/>
                    <Element libField="RawINT8" eventField="Telem-NumSamples"/>
                </fields>
            </reportHeaderFormat>
            <sampleHeaderFormat>
                <fields>
                    <Element libField="Raw" eventField="Telem-Latitude"/>
                    <Element libField="Raw" eventField="Telem-Longitude"/>
                    <Element libField="Raw" eventField="Telem-TimeSec"/>
                    <Element libField="Raw" eventField="C3Event_Header"/>
                </fields>
            </sampleHeaderFormat>
        </second>
    </Element>
    <Element>
        <first value="1"/>
        <second maxSamples="7" targetAddress="2">
            <reportHeaderFormat>
                <fields>
                    <Element libField="Header91"/>
                    <Element libField="Raw" eventField="Telem-LengthToFollow"/>
                    <Element libField="Raw" eventField="Telem-PayloadType"/>
                    <Element libField="Raw" eventField="Telem-BoardID"/>
                    <Element libField="Raw" eventField="Telem-TaskID"/>
                    <Element libField="FC9BinaryNormal"/>
                    <Element libField="RawINT8" eventField="Telem-NumSamples"/>
                </fields>
            </reportHeaderFormat>
            <sampleHeaderFormat>
                <fields>
                    <Element libField="Raw" eventField="Telem-Latitude"/>
                    <Element libField="Raw" eventField="Telem-Longitude"/>
                    <Element libField="Raw" eventField="Telem-TimeSec"/>
                    <Element libField="Raw" eventField="C3Event_CompressedData"/>
                </fields>
            </sampleHeaderFormat>
        </second>
    </Element>
</reportParams>
</second>
</Element>
<Element>
    <first boardID="0xFF" taskID="0x10"/>
    <second>
        <key boardID="0xFF" taskID="0x10"/>
        <payloadType value="0x00000080"/>
        <eventNameToFormatCode>
            <Element>
                <first value="ADCPSampleHeaders"/>
                <second value="0"/>
            </Element>
            <Element>
                <first value="ADCPSampleB"/>
                <second value="1"/>
            </Element>
            <Element>
                <first value="ADCPSampleC2"/>

```

```

        <second value="2"/>
    </Element>
    <Element>
        <first value="ADCPsampleC4"/>
        <second value="3"/>
    </Element>
</eventNameToFormatCode>
<reportParams>
    <Element>
        <first value="0"/>
        <second maxSamples="1" targetAddress="2">
            <reportHeaderFormat>
                <fields>
                    <Element libField="Header91"/>
                    <Element libField="Raw" eventField="Telem-LengthToFollow"/>
                    <Element libField="Raw" eventField="Telem-PayloadType"/>
                    <Element libField="Raw" eventField="Telem-BoardID"/>
                    <Element libField="Raw" eventField="Telem-TaskID"/>
                    <Element libField="FC2BinaryNormal"/>
                    <Element libField="RawINT8" eventField="Telem-NumSamples"/>
                </fields>
            </reportHeaderFormat>
            <sampleHeaderFormat>
                <fields>
                    <Element libField="Raw" eventField="Telem-Latitude"/>
                    <Element libField="Raw" eventField="Telem-Longitude"/>
                    <Element libField="Raw" eventField="Telem-TimeSec"/>
                    <Element libField="Raw" eventField="ADCPEvent-Binary"/>
                </fields>
            </sampleHeaderFormat>
        </second>
    </Element>
    <Element>
        <first value="1"/>
        <second maxSamples="1" targetAddress="2">
            <reportHeaderFormat>
                <fields>
                    <Element libField="Header91"/>
                    <Element libField="Raw" eventField="Telem-LengthToFollow"/>
                    <Element libField="Raw" eventField="Telem-PayloadType"/>
                    <Element libField="Raw" eventField="Telem-BoardID"/>
                    <Element libField="Raw" eventField="Telem-TaskID"/>
                    <Element libField="FC3BinaryNormal"/>
                    <Element libField="RawINT8" eventField="Telem-NumSamples"/>
                </fields>
            </reportHeaderFormat>
            <sampleHeaderFormat>
                <fields>
                    <Element libField="Raw" eventField="Telem-Latitude"/>
                    <Element libField="Raw" eventField="Telem-Longitude"/>
                    <Element libField="Raw" eventField="Telem-TimeSec"/>
                    <Element libField="Raw" eventField="ADCPEvent-Binary"/>
                </fields>
            </sampleHeaderFormat>
        </second>
    </Element>
    <Element>
        <first value="2"/>
        <second maxSamples="1" targetAddress="2">
            <reportHeaderFormat>
                <fields>
                    <Element libField="Header91"/>
                    <Element libField="Raw" eventField="Telem-LengthToFollow"/>
                    <Element libField="Raw" eventField="Telem-PayloadType"/>
                    <Element libField="Raw" eventField="Telem-BoardID"/>
                    <Element libField="Raw" eventField="Telem-TaskID"/>
                    <Element libField="FC4BinaryNormal"/>
                    <Element libField="RawINT8" eventField="Telem-NumSamples"/>
                </fields>
            </reportHeaderFormat>
            <sampleHeaderFormat>

```

```

        <fields>
        <Element libField="Raw" eventField="Telem-Latitude"/>
        <Element libField="Raw" eventField="Telem-Longitude"/>
        <Element libField="Raw" eventField="Telem-TimeSec"/>
        <Element libField="Raw" eventField="ADCPEvent-Binary"/>
        </fields>
    </sampleHeaderFormat>
</second>
</Element>
<Element>
    <first value="3"/>
    <second maxSamples="1" targetAddress="2">
        <reportHeaderFormat>
            <fields>
            <Element libField="Header91"/>
            <Element libField="Raw" eventField="Telem-LengthToFollow"/>
            <Element libField="Raw" eventField="Telem-PayloadType"/>
            <Element libField="Raw" eventField="Telem-BoardID"/>
            <Element libField="Raw" eventField="Telem-TaskID"/>
            <Element libField="FC5BinaryNormal"/>
            <Element libField="RawINT8" eventField="Telem-NumSamples"/>
            </fields>
        </reportHeaderFormat>
        <sampleHeaderFormat>
            <fields>
            <Element libField="Raw" eventField="Telem-Latitude"/>
            <Element libField="Raw" eventField="Telem-Longitude"/>
            <Element libField="Raw" eventField="Telem-TimeSec"/>
            <Element libField="Raw" eventField="ADCPEvent-Binary"/>
            </fields>
        </sampleHeaderFormat>
    </second>
</Element>
</reportParams>
</second>
</Element>
<Element>
    <first boardID="0xFF" taskID="0x40"/>
    <second>
        <key boardID="0xFF" taskID="0x40"/>
        <payloadType value="0x000000FE"/>
        <eventNameToFormatCode>
            <Element>
                <first value="LWTelem1"/>
                <second value="0"/>
            </Element>
        </eventNameToFormatCode>
        <reportParams>
            <Element>
                <first value="0"/>
                <second maxSamples="1" targetAddress="2">
                    <reportHeaderFormat>
                        <fields>
                        <Element libField="Header91"/>
                        <Element libField="Raw" eventField="Telem-LengthToFollow"/>
                        <Element libField="Raw" eventField="Telem-PayloadType"/>
                        <Element libField="Raw" eventField="Telem-BoardID"/>
                        <Element libField="Raw" eventField="Telem-TaskID"/>
                        <Element libField="FC0ASCIINormal"/>
                        <Element libField="RawINT8" eventField="Telem-NumSamples"/>
                        </fields>
                    </reportHeaderFormat>
                    <sampleHeaderFormat>
                        <fields>
                        <Element libField="Raw" eventField="Telem-Latitude"/>
                        <Element libField="Raw" eventField="Telem-Longitude"/>
                        <Element libField="Raw" eventField="Telem-TimeSec"/>
                        <Element libField="Raw" eventField="StringMessage"/>
                        </fields>
                    </sampleHeaderFormat>
                </second>
            </Element>
        </reportParams>
    </second>
</Element>

```

```

        </Element>
    </reportParams>
</second>
</Element>
<Element>
    <first boardID="0xFF" taskID="0x18"/>
    <second>
        <key boardID="0xFF" taskID="0x18"/>
        <payloadType value="0x00000083"/>
        <eventNameToFormatCode>
            <Element>
                <first value="WHCycleReport"/>
                <second value="0"/>
            </Element>
        </eventNameToFormatCode>
        <reportParams>
            <Element>
                <first value="0"/>
                <second maxSamples="1" targetAddress="2">
                    <reportHeaderFormat>
                        <fields>
                            <Element libField="Header91"/>
                            <Element libField="Raw" eventField="Telem-LengthToFollow"/>
                            <Element libField="Raw" eventField="Telem-PayloadType"/>
                            <Element libField="Raw" eventField="Telem-BoardID"/>
                            <Element libField="Raw" eventField="Telem-TaskID"/>
                            <Element libField="FC0ASCIINormal"/>
                            <Element libField="RawINT8" eventField="Telem-NumSamples"/>
                        </fields>
                    </reportHeaderFormat>
                    <sampleHeaderFormat>
                        <fields>
                            <Element libField="Raw" eventField="Telem-Latitude"/>
                            <Element libField="Raw" eventField="Telem-Longitude"/>
                            <Element libField="Raw" eventField="Telem-TimeSec"/>
                            <Element libField="Raw" eventField="StringEvent-String"/>
                        </fields>
                    </sampleHeaderFormat>
                </second>
            </Element>
        </reportParams>
    </second>
</Element>
<Element>
    <first boardID="0xFF" taskID="0x19"/>
    <second>
        <key boardID="0xFF" taskID="0x19"/>
        <payloadType value="0x00000087"/>
        <eventNameToFormatCode>
            <Element>
                <first value="LISSTSummaryDataEvent"/>
                <second value="0"/>
            </Element>
        </eventNameToFormatCode>
        <reportParams>
            <Element>
                <first value="0"/>
                <second maxSamples="5" targetAddress="2">
                    <reportHeaderFormat>
                        <fields>
                            <Element libField="Header91"/>
                            <Element libField="Raw" eventField="Telem-LengthToFollow"/>
                            <Element libField="Raw" eventField="Telem-PayloadType"/>
                            <Element libField="Raw" eventField="Telem-BoardID"/>
                            <Element libField="Raw" eventField="Telem-TaskID"/>
                            <Element libField="FC0BinaryNormal"/>
                            <Element libField="RawINT8" eventField="Telem-NumSamples"/>
                        </fields>
                    </reportHeaderFormat>
                    <sampleHeaderFormat>
                        <fields>

```

```

        <Element libField="Raw" eventField="Telem-Latitude"/>
        <Element libField="Raw" eventField="Telem-Longitude"/>
        <Element libField="Raw" eventField="Telem-TimeSec"/>
        <Element libField="Raw"
eventField="LISSTSummaryDataEvent_Volume"/>
        <Element libField="Raw"
eventField="LISSTSummaryDataEvent_Size"/>
        <Element libField="Raw"
eventField="LISSTSummaryDataEvent_SampleCount"/>
        <Element libField="Raw"
eventField="LISSTSummaryDataEvent_SerialNumber"/>
        <Element libField="Raw"
eventField="LISSTSummaryDataEvent_BackgroundFileDateLength"/>
        <Element libField="Raw"
eventField="LISSTSummaryDataEvent_BackgroundFileDate"/>
    </fields>
</sampleHeaderFormat>
</second>
</Element>
</reportParams>
</second>
</Element>
<Element>
    <first boardID="0xFF" taskID="0x1D"/>
    <second>
        <key boardID="0xFF" taskID="0x1D"/>
        <payloadType value="0x0000008C"/>
        <eventNameToFormatCode>
            <Element>
                <first value="MagnetometerRawData"/>
                <second value="0"/>
            </Element>
        </eventNameToFormatCode>
        <reportParams>
            <Element>
                <first value="0"/>
                <second maxSamples="2" targetAddress="2">
                    <reportHeaderFormat>
                        <fields>
                            <Element libField="Header91"/>
                            <Element libField="Raw" eventField="Telem-LengthToFollow"/>
                            <Element libField="Raw" eventField="Telem-PayloadType"/>
                            <Element libField="Raw" eventField="Telem-BoardID"/>
                            <Element libField="Raw" eventField="Telem-TaskID"/>
                            <Element libField="FC1BinaryNormal"/>
                            <Element libField="RawINT8" eventField="Telem-NumSamples"/>
                        </fields>
                    </reportHeaderFormat>
                    <sampleHeaderFormat>
                        <fields>
                            <Element libField="Raw" eventField="Telem-Latitude"/>
                            <Element libField="Raw" eventField="Telem-Longitude"/>
                            <Element libField="Raw" eventField="Telem-TimeSec"/>
                            <Element libField="Raw" eventField="Filtered Mag Field (nT)"/>
                            <Element libField="Raw" eventField="Date YY.JJJ"/>
                            <Element libField="Raw" eventField="Time HH:MM:SS.s"/>
                            <Element libField="Raw" eventField="Mag Field (nT)"/>
                            <Element libField="Raw" eventField="Signal Strength"/>
                            <Element libField="Raw" eventField="Depth (m)"/>
                            <Element libField="Raw" eventField="Leak"/>
                            <Element libField="Raw" eventField="Measurement Time (ms)"/>
                            <Element libField="Raw" eventField="Signal Quality"/>
                        </fields>
                    </sampleHeaderFormat>
                </second>
            </Element>
        </reportParams>
    </second>
</Element>
<Element>
    <first boardID="0xFF" taskID="0x1E"/>

```

```

<second>
  <key boardID="0xFF" taskID="0x1E"/>
  <payloadType value="0x00000091"/>
  <eventNameToFormatCode>
    <Element>
      <first value="VemcoVM4Message"/>
      <second value="0"/>
    </Element>
    <Element>
      <first value="VemcoVM4Realtime"/>
      <second value="1"/>
    </Element>
    <Element>
      <first value="VemcoVM4LocalHealth"/>
      <second value="2"/>
    </Element>
    <Element>
      <first value="VemcoVM4RemoteHealth"/>
      <second value="3"/>
    </Element>
    <Element>
      <first value="VemcoVM4DailyLocalHealth"/>
      <second value="4"/>
    </Element>
    <Element>
      <first value="VemcoVM4DailyRemoteHealth"/>
      <second value="5"/>
    </Element>
  </eventNameToFormatCode>
  <reportParams>
    <Element>
      <first value="0"/>
      <second maxSamples="1" targetAddress="2">
        <reportHeaderFormat>
          <fields>
            <Element libField="Header91"/>
            <Element libField="Raw" eventField="Telem-LengthToFollow"/>
            <Element libField="Raw" eventField="Telem-PayloadType"/>
            <Element libField="Raw" eventField="Telem-BoardID"/>
            <Element libField="Raw" eventField="Telem-TaskID"/>
            <Element libField="FC0ASCIINormal"/>
            <Element libField="RawINT8" eventField="Telem-NumSamples"/>
          </fields>
        </reportHeaderFormat>
        <sampleHeaderFormat>
          <fields>
            <Element libField="Raw" eventField="Telem-Latitude"/>
            <Element libField="Raw" eventField="Telem-Longitude"/>
            <Element libField="Raw" eventField="Telem-TimeSec"/>
            <Element libField="Raw" eventField="VemcoVM4Event_Header"/>
          </fields>
        </sampleHeaderFormat>
      </second>
    </Element>
    <Element>
      <first value="1"/>
      <second maxSamples="8" maxRetainTime="300" targetAddress="2">
        <reportHeaderFormat>
          <fields>
            <Element libField="Header91"/>
            <Element libField="Raw" eventField="Telem-LengthToFollow"/>
            <Element libField="Raw" eventField="Telem-PayloadType"/>
            <Element libField="Raw" eventField="Telem-BoardID"/>
            <Element libField="Raw" eventField="Telem-TaskID"/>
            <Element libField="FC1BinaryNormal"/>
            <Element libField="RawINT8" eventField="Telem-NumSamples"/>
          </fields>
        </reportHeaderFormat>
        <sampleHeaderFormat>
          <fields>
            <Element libField="Raw" eventField="Telem-Latitude"/>

```

```
<Element libField="Raw" eventField="Telem-Longitude"/>
<Element libField="Raw" eventField="Telem-TimeSec"/>
<Element libField="Raw" eventField="VemcoVM4Event_Realtime"/>
</fields>
</sampleHeaderFormat>
</second>
</Element>
<Element>
  <first value="2"/>
  <second maxSamples="1" targetAddress="2">
    <reportHeaderFormat>
      <fields>
        <Element libField="Header91"/>
        <Element libField="Raw" eventField="Telem-LengthToFollow"/>
        <Element libField="Raw" eventField="Telem-PayloadType"/>
        <Element libField="Raw" eventField="Telem-BoardID"/>
        <Element libField="Raw" eventField="Telem-TaskID"/>
        <Element libField="FC2BinaryNormal"/>
        <Element libField="RawINT8" eventField="Telem-NumSamples"/>
      </fields>
    </reportHeaderFormat>
    <sampleHeaderFormat>
      <fields>
        <Element libField="Raw" eventField="Telem-Latitude"/>
        <Element libField="Raw" eventField="Telem-Longitude"/>
        <Element libField="Raw" eventField="Telem-TimeSec"/>
        <Element libField="Raw" eventField="VemcoVM4Event_Health"/>
      </fields>
    </sampleHeaderFormat>
  </second>
</Element>
<Element>
  <first value="3"/>
  <second maxSamples="1" targetAddress="2">
    <reportHeaderFormat>
      <fields>
        <Element libField="Header91"/>
        <Element libField="Raw" eventField="Telem-LengthToFollow"/>
        <Element libField="Raw" eventField="Telem-PayloadType"/>
        <Element libField="Raw" eventField="Telem-BoardID"/>
        <Element libField="Raw" eventField="Telem-TaskID"/>
        <Element libField="FC3BinaryNormal"/>
        <Element libField="RawINT8" eventField="Telem-NumSamples"/>
      </fields>
    </reportHeaderFormat>
    <sampleHeaderFormat>
      <fields>
        <Element libField="Raw" eventField="Telem-Latitude"/>
        <Element libField="Raw" eventField="Telem-Longitude"/>
        <Element libField="Raw" eventField="Telem-TimeSec"/>
        <Element libField="Raw" eventField="VemcoVM4Event_Health"/>
      </fields>
    </sampleHeaderFormat>
  </second>
</Element>
<Element>
  <first value="4"/>
  <second maxSamples="1" targetAddress="2">
    <reportHeaderFormat>
      <fields>
        <Element libField="Header91"/>
        <Element libField="Raw" eventField="Telem-LengthToFollow"/>
        <Element libField="Raw" eventField="Telem-PayloadType"/>
        <Element libField="Raw" eventField="Telem-BoardID"/>
        <Element libField="Raw" eventField="Telem-TaskID"/>
        <Element libField="FC4BinaryNormal"/>
        <Element libField="RawINT8" eventField="Telem-NumSamples"/>
      </fields>
    </reportHeaderFormat>
    <sampleHeaderFormat>
      <fields>
```

```

        <Element libField="Raw" eventField="Telem-Latitude"/>
        <Element libField="Raw" eventField="Telem-Longitude"/>
        <Element libField="Raw" eventField="Telem-TimeSec"/>
        <Element libField="Raw" eventField="VemcoVM4Event_Health"/>
    </fields>
</sampleHeaderFormat>
</second>
</Element>
<Element>
    <first value="5"/>
    <second maxSamples="1" targetAddress="2">
        <reportHeaderFormat>
            <fields>
                <Element libField="Header91"/>
                <Element libField="Raw" eventField="Telem-LengthToFollow"/>
                <Element libField="Raw" eventField="Telem-PayloadType"/>
                <Element libField="Raw" eventField="Telem-BoardID"/>
                <Element libField="Raw" eventField="Telem-TaskID"/>
                <Element libField="FC5BinaryNormal"/>
                <Element libField="RawINT8" eventField="Telem-NumSamples"/>
            </fields>
        </reportHeaderFormat>
        <sampleHeaderFormat>
            <fields>
                <Element libField="Raw" eventField="Telem-Latitude"/>
                <Element libField="Raw" eventField="Telem-Longitude"/>
                <Element libField="Raw" eventField="Telem-TimeSec"/>
                <Element libField="Raw" eventField="VemcoVM4Event_Health"/>
            </fields>
        </sampleHeaderFormat>
    </second>
</Element>
</reportParams>
</second>
</Element>
<Element>
    <first boardID="0xFF" taskID="0x25"/>
    <second>
        <key boardID="0xFF" taskID="0x25"/>
        <payloadType value="0x00000093"/>
        <eventNameToFormatCode>
            <Element>
                <first value="GPSWavesTelem"/>
                <second value="0"/>
            </Element>
        </eventNameToFormatCode>
        <reportParams>
            <Element>
                <first value="0"/>
                <second maxSamples="1" targetAddress="2">
                    <reportHeaderFormat>
                        <fields>
                            <Element libField="Header91"/>
                            <Element libField="Raw" eventField="Telem-LengthToFollow"/>
                            <Element libField="Raw" eventField="Telem-PayloadType"/>
                            <Element libField="Raw" eventField="Telem-BoardID"/>
                            <Element libField="Raw" eventField="Telem-TaskID"/>
                            <Element libField="FC0BinaryNormal"/>
                            <Element libField="RawINT8" eventField="Telem-NumSamples"/>
                        </fields>
                    </reportHeaderFormat>
                    <sampleHeaderFormat>
                        <fields>
                            <Element libField="Raw" eventField="Telem-Latitude"/>
                            <Element libField="Raw" eventField="Telem-Longitude"/>
                            <Element libField="Raw" eventField="Telem-TimeSec"/>
                            <Element libField="Raw" eventField="GPSWavesTelemEvent_Hs"/>
                            <Element libField="Raw" eventField="GPSWavesTelemEvent_Tp"/>
                            <Element libField="Raw" eventField="GPSWavesTelemEvent_Dp"/>
                            <Element libField="Raw"
eventField="GPSWavesTelemEvent_Samples"/>
                        </fields>
                    </sampleHeaderFormat>
                </second>
            </Element>
        </reportParams>
    </second>
</Element>

```

```

        <Element libField="Raw" eventField="GPSWavesTelemEvent_Fs"/>
        <Element libField="Raw"
eventField="GPSWavesTelemEvent_SampleGaps"/>
        <Element libField="Raw" eventField="GPSWavesTelemEvent_Ta"/>
    </fields>
</sampleHeaderFormat>
</second>
</Element>
</reportParams>
</second>
</Element>
<Element>
    <first boardID="0xFF" taskID="0x26"/>
    <second>
        <key boardID="0xFF" taskID="0x26"/>
        <payloadType value="0x00000070"/>
        <eventNameToFormatCode>
            <Element>
                <first value="SBGPCTDSample"/>
                <second value="8"/>
            </Element>
            <Element>
                <first value="SBGPCTDSampleWithD0"/>
                <second value="9"/>
            </Element>
        </eventNameToFormatCode>
        <reportParams>
            <Element>
                <first value="8"/>
                <second maxSamples="1" targetAddress="2">
                    <reportHeaderFormat>
                        <fields>
                            <Element libField="Header91"/>
                            <Element libField="Raw" eventField="Telem-LengthToFollow"/>
                            <Element libField="Raw" eventField="Telem-PayloadType"/>
                            <Element libField="Raw" eventField="Telem-BoardID"/>
                            <Element libField="Raw" eventField="Telem-TaskID"/>
                            <Element libField="FC8BinaryNormal"/>
                            <Element libField="RawINT8" eventField="Telem-NumSamples"/>
                        </fields>
                    </reportHeaderFormat>
                    <sampleHeaderFormat>
                        <fields>
                            <Element libField="Raw" eventField="Telem-Latitude"/>
                            <Element libField="Raw" eventField="Telem-Longitude"/>
                            <Element libField="Raw" eventField="Telem-TimeSec"/>
                            <Element libField="Raw" eventField="SBGPCTDSample_RawPressure"/>
                            <Element libField="Raw"
eventField="SBGPCTDSample_RawTemperature"/>
                            <Element libField="Raw"
eventField="SBGPCTDSample_RawConductivity"/>
                        </fields>
                    </sampleHeaderFormat>
                </second>
            </Element>
        </reportParams>
        <first value="9"/>
        <second maxSamples="1" targetAddress="2">
            <reportHeaderFormat>
                <fields>
                    <Element libField="Header91"/>
                    <Element libField="Raw" eventField="Telem-LengthToFollow"/>
                    <Element libField="Raw" eventField="Telem-PayloadType"/>
                    <Element libField="Raw" eventField="Telem-BoardID"/>
                    <Element libField="Raw" eventField="Telem-TaskID"/>
                    <Element libField="FC9BinaryNormal"/>
                    <Element libField="RawINT8" eventField="Telem-NumSamples"/>
                </fields>
            </reportHeaderFormat>
            <sampleHeaderFormat>
                <fields>

```

Page 109 of 111

```
</Element>
<Element>
  <key value="FC5BinaryNormal"/>
  <value content="CA_LITERAL">
    <literal type="UINT8" value="0x05"/>
  </value>
</Element>
<Element>
  <key value="FC6BinaryNormal"/>
  <value content="CA_LITERAL">
    <literal type="UINT8" value="0x06"/>
  </value>
</Element>
<Element>
  <key value="FC7BinaryNormal"/>
  <value content="CA_LITERAL">
    <literal type="UINT8" value="0x07"/>
  </value>
</Element>
<Element>
  <key value="FC8BinaryNormal"/>
  <value content="CA_LITERAL">
    <literal type="UINT8" value="0x08"/>
  </value>
</Element>
<Element>
  <key value="FC9BinaryNormal"/>
  <value content="CA_LITERAL">
    <literal type="UINT8" value="0x09"/>
  </value>
</Element>
<Element>
  <key value="RawINT8"/>
  <value content="CA_LENGTH1"/>
</Element>
</lib>
</fieldLibrary>
</root>
```

History

date	changes made	author
08/06/2014	Additions for 2.3.0 release	Mike Cookson