

PopotoModem

User's Guide for PMM6081 M2000/M6000 DB100-6

delResearch LLC

Document Versions

Version	Purpose	Author	Sw Ver	Date
1.01	Initial draft	Jim DellaMorte		9/28/2018
1.02	Updated Document SSB	John DellaMorte		2/2/2019
1.03	Added many new commands	John DellaMorte		1/15/2020
1.04	Fixed page numbers	John DellaMorte		2/26/2020
1.05	Added commands	John DellaMorte	2.06	5/28/2020
1.06	Added commands	J DellaMorte	2.2x	9/21/2020
1.07	Added commands	J DellaMorte	2.7x	3/8/2021
1.10	Major Overhaul of document	J DellaMorte	2.7x	4/28/2021
1.20	Add Interface board and Battery Mux	J DellaMorte	3.x	9/2022
1.30	Add PMM6081 and wave file record	J DellaMorte	3.x	1/2024

Contents

1	Getting Started	17
1.1	Introduction	17
1.2	Delivery Options	17
1.2.1	OEM Boardset	17
1.2.2	Lab Evaluation Platform	17
1.2.3	Enclosed Modem	17
1.2.4	Deckbox	17
1.3	Required equipment	18
1.4	Bench Testing	18
1.4.1	What is an Air Test	18
1.4.2	RS-422 UART connection	18
1.4.3	Running the application	19
1.4.4	Checking the version number	19
1.4.5	Displaying Help	20
1.4.6	Sending a Test Message	20
1.4.7	Sending an Arbitrary Message	20
1.4.8	Addressing a particular modem	20
1.4.9	Setting the data Rate of the Payload	21
1.4.10	Telnet Chat Operation	22
1.4.11	Sending a Range Command	23
2	OEM Modems	25
2.1	OEM Modem System	25
2.1.1	Analog Board	25
2.1.2	Digital Board	25
2.1.3	micro SD Card	26
2.1.4	Heat Sink/Mounting Tray	27
3	Lab Evaluation Platform	29
3.1	Introduction	29
3.2	What's in the Box	29
3.3	Setup Overview	29
4	DB100 Deckbox	31
4.1	Description	31
4.2	Deckbox Models	32
4.3	Dunking Transducer	32

4.4	Front Panel connections	32
4.4.1	Transducer	32
4.4.2	Transducer Connector Specifications on the DB100 Deck-box	32
4.4.3	Models DB100-3, DB100-5, and DB100-5-2	32
4.4.4	Model DB100-6	33
4.4.5	USB	33
4.4.6	Ethernet	33
4.4.7	Charging	33
4.4.8	Headphone and Microphone Jack	34
4.5	Single Sideband Voice Feature of the DB100-5-SSB	34
4.6	Getting Started with the DB100	34
4.7	USB Connection	35
4.8	Initial Setup	35
4.8.1	Connecting the Transducer	35
4.8.2	Powering On	35
4.8.3	Connecting to the Modem	35
4.9	Additional Setup for Multiple Units	35
4.10	Care	36

5 Enclosed Modems

	M2000/M6000/S1000	37
5.1	Enclosed Modems	37
5.1.1	Popoto M2000	37
5.1.2	Popoto S1000	37
5.1.3	Popoto M6000	37
5.2	Connectors	37
5.3	M2000/S1000 Connectors	37
5.3.1	Connector Part Numbers	38
5.3.2	10 Pin Ethernet Option	39
5.3.3	10 Pin RS-422 Option	39
5.3.4	16 Pin Universal Option	40
5.3.5	8 Pin Ethernet Option	41
5.3.6	8 Pin RS-422 Option	41
5.3.7	8 Pin RS-232 Option	42
5.4	Lab Cable	42
5.5	Types of Lab Cables	42
5.5.1	16-Pin Lab Cable	42
5.5.2	8-Pin Lab Cables for S1000	42
5.5.3	Extension Cables	43
5.6	Power Down and Dummy Plugs	43
5.6.1	Understanding Power Down Plugs	43
5.6.2	Design and Functionality	43
5.6.3	The Power Down Plug	44
5.7	Charging	44
5.7.1	Using the Popoto Li-ion Battery Charger	45
5.8	Disassembly	45
5.8.1	Recommendation	45

5.8.2	M2000/M6000	45
5.8.3	S1000	45
6	Popoto Transducer (PMT-28)	47
6.1	Transducer	47
6.2	Transducer Configurations	47
6.2.1	Internal Wiring Configuration	47
6.2.2	External Wiring Configuration	47
6.2.3	Dunking Transducer	47
6.2.4	Transducer Performance	48
6.3	Performance Curves	48
6.3.1	Free Field Voltage Sensitivity	48
6.3.2	Transmit Voltage Response	49
6.3.3	Transmit Beam Pattern	49
7	Communicating with Popoto	51
7.1	Introduction	51
7.2	Socket based JSON	51
7.2.1	Highlevel Description of Popoto API Sockets	51
7.2.2	Introduction JSON Messages	52
7.2.3	Commands	52
7.2.4	The Keyword Return Values	53
7.2.5	System Level Variables	53
7.3	Facilitating JSON messages	53
8	System Connections	55
8.1	RS-422 4 wire serial	55
8.1.1	Reasons to use it	55
8.1.2	Reasons to avoid it	55
8.2	RS-232 Uart	55
8.2.1	Reasons to use it	55
8.2.2	Reasons to avoid it	56
8.3	10/100BaseT	56
8.3.1	Reasons to Use it	56
8.3.2	Reasons to avoid it	56
8.4	TTL (3.3V) UART	56
8.4.1	Reasons to use it	56
8.4.2	Reasons to avoid it	56
8.5	Modes of operations	56
8.5.1	Local pshell	57
8.5.2	Remote pshell	57
8.5.3	Matlab™	57
8.5.4	Custom interfaces	58
8.6	Sleep and Power Down	59

9 Pshell	61
9.1 Modes of operation	61
9.2 Requirements for running	61
9.3 Invoking pshell	61
9.3.1 The pshell.init file	61
9.3.2 Invoking pshell from a linux prompt	62
9.4 Invoking commands	62
9.4.1 Help	62
9.4.2 Tab Completion	62
9.4.3 Commands	62
9.5 Extending the pshell	62
9.6 Scheduling Transmissions	63
9.6.1 Streaming Mode Operation for Popoto Modem	64
10 Popoto Modulation Data Rates and Payload Modes	67
10.1 PayloadMode Enumeration	67
10.2 Coding Structure	68
10.2.1 Coherent Modes (BPSK & QPSK)	68
10.2.2 Uncoded Mode (QPSK_10240bps)	68
10.2.3 FH-FSK Mode	68
10.3 Recommended Modulation Use	68
10.4 Payload Mode Summary	69
10.5 Measured SNR Waterfall Thresholds	69
10.6 Mode Selection Summary	69
11 Single Side Band Voice Operation	71
12 Single Side Band Voice Operation	73
12.1 Overview	73
12.2 SSB Transmitter	73
12.2.1 The ssbtx command	73
12.2.2 Adjustment of transmit power	74
12.2.3 Peak Envelope Power	74
12.2.4 PTT keying of the transmitter	74
12.2.5 Transmitter Vox	75
12.3 SSB Receiver	75
12.3.1 Squelch	75
12.3.2 Noise Reduction and AGC	75
12.4 Return to data mode	76
12.5 SSB Controllable parameters	76
13 Janus Operation	77
13.1 Janus Overview	77
13.2 Janus Bitstream	77
13.3 Popoto Modem Implementation of Janus	78
13.3.1 Class User Id: 000 Emergency	78

13.3.2	Class User Id: 002 Underwater AIS	81
13.3.3	Class User Id: 016 NATO JANUS reference	82
14	Phorcys Operation	83
14.1	Phorcys System Configuration	83
14.1.1	Transmission Primitives	83
14.1.2	System Variable Summary	83
14.1.3	Short-Payload (SP) Waveform Specification	84
14.1.4	Long-Payload (LP) Waveform Specification	84
14.1.5	Distinguishing Phorcys from Popoto Modulation on Re- ception	85
15	OEM Interface Description	87
15.1	Popoto Digital Interface	87
15.1.1	Overview	87
15.1.2	PDI Hardware Components	87
15.1.3	Electrical Connections	89
15.1.4	Digital Interfaces	90
15.2	PMM6081 Specific Interfaces	95
15.3	PMM6081 Specific Interfaces	95
15.3.1	Power	95
15.3.2	Analog Interfaces	96
15.3.3	Analog Board	98
16	Battery Multiplexer	103
16.1	Battery Multiplexer	103
16.1.1	Overview	103
16.1.2	Power Connectors	103
16.1.3	Charging Connectors	106
17	Popoto Interface Board	107
17.1	Popoto Interface Board	107
17.1.1	Overview	107
17.1.2	PDI Connector J10	107
17.1.3	USB Port (J2)	107
17.1.4	Ethernet Port (J1)	107
17.1.5	Switch SW1 and Jumper J9	108
17.1.6	PWRDN LED (J8)	108
17.1.7	SSB Connections	108
18	Upgrading the Firmware	113
18.1	Introduction	113
18.1.1	Details on how to update the firmware	113
18.2	Upload Procedure	113
18.3	Updating Image using UUU	116
18.3.1	Updating with fast uuu	117
19	Network Settings	119
19.1	Adjusting Network Settings on Popoto Debian	119

20 Diagnostics	121
20.1 Popoto log	121
20.1.1 Introduction	121
20.1.2 Location	121
20.1.3 Logging Levels	122
20.1.4 MSM Logs	122
20.2 PCM Logging	122
20.2.1 Introduction	122
20.2.2 Socket based PCMLogs	123
20.2.3 Target File based PCM Logs	124
20.2.4 Notes	124
20.3 pshell Logging	125
21 Automatic PCM Error Logging	127
21.1 Overview	127
21.2 How to Enable	127
21.3 Logging Details	127
22 Pshell Command Reference	129
Rx	130
chat	131
configure	132
connect	133
datamode	134
deepsleep	135
disablemsmlog	136
disconnect	137
download	138
enablemsmlog	139
exit	140
getEXPI	141
getIP	142
getPEP	143
getclock	144
getvaluef	145
getvaluei	146
getverbosity	147
ls	148
mips	149
multiping	150
netplay	151
netrec	152
ping	153
playstart	154
playstop	155
powerdown	156
q	157
quit	158

range	159
recordstart	160
recordstop	161
remote	162
setEXP0	163
setRate10240	164
setRate1280	165
setRate2560	166
setRate5120	167
setRate640	168
setRate80	169
setTerminalMode	170
setcarrier	171
setcarrier25	172
setcarrier30	173
setclock	174
setgainmode	175
setvaluef	176
setvaluei	177
setverbosity	178
sleep	179
ssb	180
ssbtx	181
startrx	182
transmit	183
transmitJSON	184
transmitJSONFiles	185
unq	186
upload	187
version	188

23 Popoto Variables 189

APP_CycleCount	190
APP_CycleCountReset	191
APP_ModemSMAOut	192
APP_SocketBasedPCM	193
BBAND_DownCarrier	194
BBAND_OutputScale	195
BBAND_UpCarrier	196
BatteryCharge	197
BatteryCurrent	198
BatteryVoltage	199
Carrier	200
CarrierTxMode	201
ChannelIR	202
ConsolePacketBytes	203
ConsoleTimeoutMS	204
DOWNCONVERT_Carrier	205

DataPortMode	206
DepartureDelay_samples	207
DopplerDisable	208
DopplerEnable	209
FHDEMODO_DetectThresholdDB	210
FHDEMODO_HopSeqLen	211
FHMOD_HopSeqLen	212
FHMOD_NumChipReps	213
GainAdjustMode	214
HP_FilterEnable	215
Headerless	216
InBandNoiseEnergy	217
InbandEnergy	218
LedEnable	219
LocalID	220
LoggingLevel	221
MODEM_Enable	222
PSK_BnTaps	223
PSK_Constellation	224
PSK_FnTaps	225
PSK_PDSNR	226
PSK_PLL	227
PSK_Taps	228
PayloadMode	229
PeakEnvelopePower	230
PlayMode	231
RNG_SpeedOfSound	232
RNG_TA_DelayMs	233
RangeTimeout_mS	234
RecordMode	235
RemoteID	236
ReportErroredBytes	237
RxEnable	238
RxScramblerMode	239
SNR	240
SSB_Enable	241
SSB_NREnable	242
SSB_SetPTT	243
SSB_SqLevel	244
SSB_Txpower	245
SSB_UseGPIO	246
SSB_Volume	247
SSB_VxHang	248
SSB_VxLevel	249
SSB_VxMode	250
SSB_carrier	251
SSB_sideband	252
SignalEnergy	253

StreamingTxLen	254
SystemMode	255
TCPecho	256
Temp_Ambient	257
TxChirpMode	258
TxEnable	259
TxPower	260
TxPowerWatts	261
TxTimeout_mS	262
UPCONVERT_Carrier	263
UPCONVERT_OutputScale	264
UnitTestMode	265
brdState	266
rxState	267
tpaState	268
24 TIPE	269
24.1 TIPE Commands	269
range	269
ping	271
24.2 TIPE Variables	272
echosounder:mode	272
echosounder:carrier	273
echosounder:hfmchirpbw	274
echosounder:hfmchirplength	275
echosounder:timeout	276
echosounder:speedofsound	277
transponder:mode	278
transponder:carrier	279
transponder:detecttonef	280
transponder:detecttonelength	281
transponder:turnaroundtime	282
transponder:respondtonef	283
transponder:respondtonelength	284
transponder:timeout	285
pinger:mode	286
pinger:carrier	287
pinger:pingtone	288
pinger:tonelength	289
25 Appendix A	291
25.1 The Acoustic Message Header	291
25.2 Payload Structure	292
26 Appendix B	293
26.1 Assembly Drawings	293

List of Tables

1.1	PSK Parameters	22
1.2	Chat Mode Parameters	23
4.1	Pin Assignments for DB100-3, DB100-5, and DB100-5-2	33
4.2	Pin Assignments for DB100-6	33
4.3	Pinout for Tensility Intl.: 10-02845 Connector	34
10.1	PayloadMode Summary	69
10.2	Measured SNR Waterfall Thresholds	69
12.1	SSB Control Bits	74
12.2	SSB Pinout	75
12.3	SSB Control Bits	76
14.2	PHORCYsv0 – Short Payload (SP) waveform specification summary.	84
14.3	PHORCYsv0 – Long Payload (LP) waveform specification summary.	84
15.1	PDI Components and Part Numbers	89
15.2	PDI Electrical Pinout	90
15.3	Extended Gigabit PDI Electrical Pinout	90
15.4	Popoto TTL UART Parts	91
15.5	Popoto 3.3V Uart Port	91
15.6	Popoto Expansion Header Parts	93
15.7	0056 Analog Board GPIO Expansion Header	93
15.8	PMM6081 Power Plug Components	95
15.9	PMM6081 Power Connector Pinout	96
15.10	J1 High Gain Hydrophone Connector Pinout, AINxP and AINxN are the positive and negative inputs for a given hydrophone channel.	97
15.11	J2 Low Gain Hydrophone Connector Pinout, AINxP and AINxN are the positive and negative inputs for a given hydrophone channel.	97
15.12	Manufacturer Part Number for iGrid Pins and Shells	98
15.13	PMM6081 Transducer Connector Pinout	100
15.14	PMM6081 Transducer Plug Parts	100

16.1	Battery Multiplexer Power In and Out Plug Components	103
16.2	Battery Mux Power Connector Pinout	103
16.3	Battery Mux Charging Port Part Numbers	106
17.1	USB Ports	107
17.2	PWRDN LED Connector Pinout	108
17.3	SMA Ports J6 and J7	108
17.4	SSB Control Port J3	108
17.5	Headphone/Microphone Connector J5	109
20.1	PCM Packet Format	123
23.1	Argument List format	189
23.2	Variable Set Return Conditions	189
25.1	Header packet format	291
25.2	Header Byte 4	292
25.3	Header Byte 5	292
25.4	Modulation Types (Mod Values)	292

List of Figures

1.1	High Level Popoto Block Diagram	19
1.2	PSK Payload Message Structure	21
2.1	The PMM6081 Analog board	26
2.2	The PMM6081 Digital board	26
2.3	The PMM5021 Mounting Tray with M4 mounting screws	27
4.1	The Popoto DB100 Deckbox	31
5.1	Pin Locations: 10 Pin and 8 Pin Connectors as viewed from the face of the male connector.	38
6.1	The Popoto 28 Khz Transducer	48
6.2	The Popoto 28 Khz Transducer Free Field Voltage Sensitivity	49
6.3	The Popoto 28 Khz Transducer Transmit Voltage Response	50
6.4	The Popoto 28 Khz Transducer Transmit Voltage Response	50
7.1	Popoto System Overview.	51
7.2	Popoto Modem Socket interfaces.	52
7.3	JSON API Interfacing to Python.	53
7.4	Popoto Modem Matlab and JSON API.	54
7.5	Popoto pshell to Popoto.py to JSON.	54
8.1	Local Pshell operation.	57
8.2	Connecting to Popoto using a remote pshell	58
8.3	Connecting to Popoto Over Matlab™.	58
8.4	Popoto Modem implementing a custom application. In this picture, a Popoto modem is configured to measure a temperature sensor, and report its measurements via the acoustic channel.	59
13.1	Janus Baseline Bit Allocation. (from the Januswiki.com webiste)	77
15.1	PDI User-Side Molex Connector. Interfacing to the PDI is accomplished with a Molex Microfit shell P/N 0430251400 and either Pre-pinned jumper wires, or Molex socket crimps.	87
15.2	Additional PDI Connection to enable Gigabit Ethernet.	88
15.3	PDI Extended Gigabit User-Side Molex Connector. By Adding the 2 additional differential pairs, the Extended Gigabit connector enables gigabit ethernet to the PMM6081.	88

15.4 PDI Schematic connections.	89
15.5 Popoto TTL UART Plug. This port allows 3.3V logic-level UART connections	91
15.6 Popoto Expansion Header. This connector allows access to I2C, SPI and General purpose I/O from the Popoto Modem.	92
15.7 Popoto 0056 Board Expansion Header. This connector allows access to I2C, and General Purpose I/O (GPIO) from the Popoto Modem.	93
15.8 PMM6081 Digital Board Connector Locations	95
15.9 PMM6081 Power Schematic.	96
15.10 PMM6081 Power Connectors and pinout.	96
15.11 The PMM6081 Analog board	98
15.12 The PMM6081 Transducer connector schematic.	99
15.13 PMM6081 Transducer connector and pinout.	99
15.14 PMM6081 Analog input schematic excerpt. This circuit conditions the input signal and is used for SSB voice input or for applications providing line level analog input.	101
15.15 PMM6081 Analog output schematic excerpt. This circuit provides a +/- 3.3V signal to the SMA output port. This signal is used for the headphones output during SMA voice mode, or for a diagnostic port or to drive an external power amplifier if needed.	102
16.1 Popoto Battery Mux Board. Power is provided by 1-5 6S Lithium Ion Batteries provided on the Molex MiniFit Jr 4 pin connectors on the left. Power is drawn from the Minifit Jr 4 port connector on the top, and charging is accomplished by connecting the charger to the 2 port microfit connector on the right side of the diagram.	104
16.2 Battery Mux: Battery in Power out Schematic.	104
16.3 PMM5021 Power Connectors and pinout.	105
16.4 Battery Mux: Charging Port Schematic. Use a constant current/-constant voltage lithium ion charger for 6S Battery Packs. (1.875A)	106
17.1 Popoto Interface Board	110
17.2 Popoto Interface Board Schematic	111
18.1 PMM6081 Switch Position used for running program command	116
18.2 PMM6081 Switch Position used for running program command	116
20.1 Format of a single PCM Log Packet. These packets are transmitted on the TCP PCM Recording socket.	123
20.2 The PCM Packets are sent one after the other to the TCP Socket or to the Target log file	124

1 Getting Started

1.1 Introduction

This section provides an overview of the various ways your Popoto Modem can be delivered and how to start using it. Each delivery option is tailored to different usage scenarios, from development and testing environments to ready-to-deploy applications.

1.2 Delivery Options

1.2.1 OEM Boardset

The OEM Boardset is designed for developers and manufacturers who wish to integrate the Popoto Modem into their own products or solutions. This option offers the most flexibility for custom applications. For detailed cabling and setup instructions, please refer to Chapter 2.

1.2.2 Lab Evaluation Platform

This delivery option includes an OEM Boardset mounted on a Lab Evaluation Platform, complete with a battery, a transducer, and an interface board for connection to a personal computer. It's ideal for quick testing and development. For setup details, see Chapter 3.

1.2.3 Enclosed Modem

The Enclosed Modem is housed in a depth-tolerant, waterproof enclosure, making it suitable for direct deployment in marine environments. Connectivity is provided through a wet-mate subsea connector. Cabling and operation information can be found in Chapter 5.

1.2.4 Deckbox

A complete modem solution housed in a Pelican case, the Deckbox option includes an external dunking transducer for immediate deployment. This option is designed for ease of use and durability. For more information on connecting and using the Deckbox, refer to Chapter 4.

1.3 Required equipment

Along with the hardware and software that comprise Popoto, it is necessary have the following equipment to facilitate the “Getting Started” procedure of this chapter.

- 12-36 Volt 5 Amp DC Power Source
- Ethernet Cable
- PC Running Ubuntu with ethernet capability
- Popoto Digital Interface

Other helpful PC software to have at the ready includes

- MATLAB
- Audacity Audio Software
- Python 3.10 or greater
- Serial Port Terminal software

1.4 Bench Testing

1.4.1 What is an Air Test

Although the Popoto modem is designed to operate acoustically in an ocean environment, it can communicate (although somewhat less reliably) in air. The acoustic energy transmitted from Modem 1 can indeed be propagated through the air for short distances and received by Modem 2. Assuming the multipath energy from sound reflection of the walls is not too damaging, this signal can be detected and demodulated. If the multipath of the room prevents detection, some careful placement of sound absorbing materials such as foam or cloth, and repositioning either the transmitter or receiver transducer until reliable communication is usually possible.

Running an air test is a good way to validate operation prior to water operation. Once reliable communication is achieved, various commands such as ranging can be exercised effectively. It should be noted that the range command will not yield accurate range estimates in air because the speed of sound in air is more than 5 times slower than the speed of sound in water. However, ranging in air is still useful for basic system checkout prior to fielding the modem in the water.

1.4.2 RS-422 UART connection

For the purpose of “Getting Started”, it is recommended that the RS-422 connection be used. This port can be accessed using the first serial port that enumerates when connected to the Popoto Interface Board’s USB port.

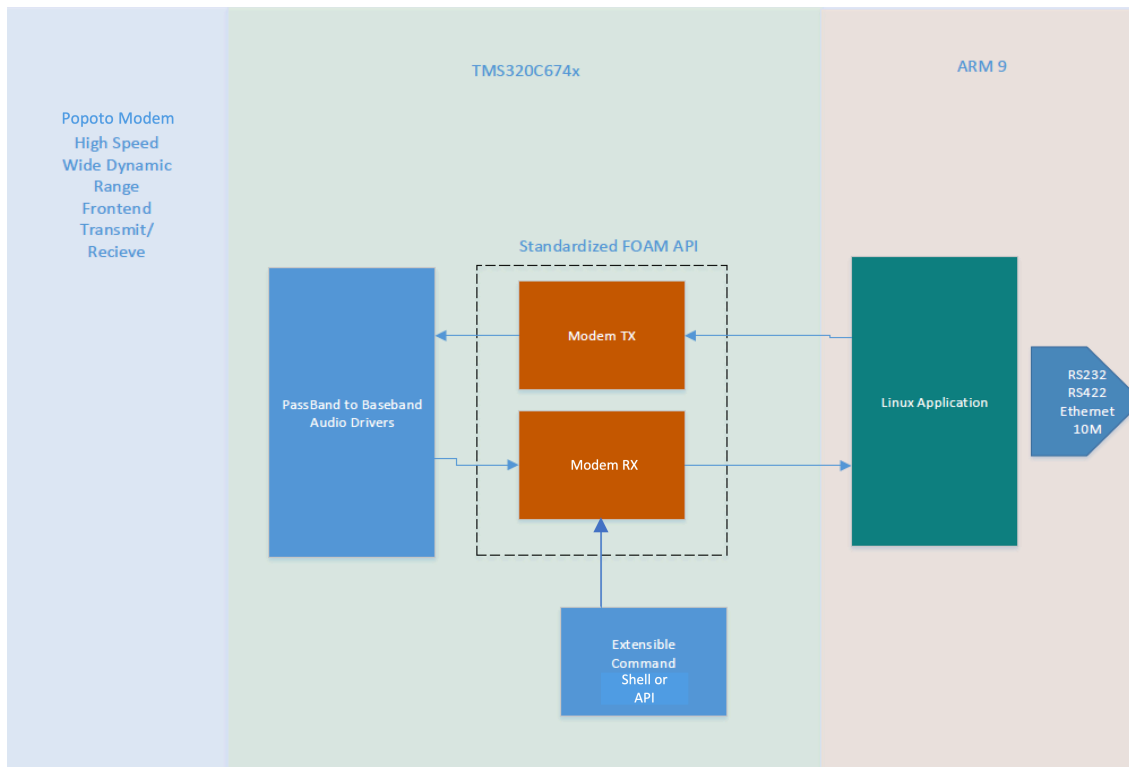


Figure 1.1: High Level Popoto Block Diagram

If the Popoto is delivered in a deckbox, this connection will be a standard USB communication port connection.

When the UART USB cable is inserted the OS will discover the new communication device. At that time open any standard serial terminal program configured for communication at 115200 bps, with no parity, 8 data bits, and 1 stop bit. About 20 seconds after power on, the user will be presented with a pshell command prompt.

1.4.3 Running the application

For this section it is assumed that we will be running air tests under a serial connected pshell.

Once the modem has been completed the boot process it is possible to connect to the Popoto by way of pshell.

1.4.4 Checking the version number

From the pshell type

```
version
```

This is the version command. Popoto will respond with current software version number and serial number of the hardware.

1.4.5 Displaying Help

To list the commands supported by the pshell, simply type

```
help
```

Popoto will respond with a list of supported pshell commands. Note that tab completion for these commands is supported.

1.4.6 Sending a Test Message

When both modems are online and connected to their pshells, it is possible to send an acoustic ping from one modem to be received by the other.

At the pshell type

```
ping 4
```

This command initiates the transmission of a test packet at about 4 watts of acoustic power. This level of power is appropriate for an air test where the transducers of units are spaced 1-2 meters apart.

While the transmission is executing you will notice a red “transmitting” led illuminate on the transmitters analog board. Once the transmission completes (3-4 seconds) the led will turn off. On the receiver pshell, there should be indication of a packet received and the both the packet and the header data should be displayed.

A message indicating ‘CRC error’ may occur at the time of transmission on the receiving Popoto instead of a ‘CRC check’ message. This occurs if the multipath of the room is adding so much interference that the demodulator cannot successfully demodulate the test packet. In such a case, reposition the transducers or pad any reflective surfaces to minimize acoustic reflection.

1.4.7 Sending an Arbitrary Message

To transmit an arbitrary message from the pshell, the transmitJSON API provides the most flexible interface. A JSON message formatted as below is passed as an argument to the transmitJSON command.

```
transmitJSON { "Payload": {"Data": [ 49,50,51,52,53,54,55,56,57,48,49,50,51,52,53,54]}}
```

sends the bytestream 49,50,51,52,53,54,55,56,57,48,49,50,51,52,53,54. to the remote modem.

1.4.8 Addressing a particular modem

Each modem has a LocalID, which is an address between 0 and 254. To address a message to a particular modem, it is necessary to know the remote modem's ID. Then, by setting the variable RemoteID, to the value of the target modem's LocalID, it is possible to address subsequent messages to the desired modem.

1.4.9 Setting the data Rate of the Payload

All packets comprised of three parts the acquisition, the header, and an optional payload. The acquisition sequence does not change for different data rates. The header is always sent at 80bps frequency hopping mode. The header contains information such as the transmit ID, intended receiver ID (broadcast ID is 255), transmit power, if there is a payload of information following, what the payload length is, and what the modulation scheme for sending the payload is.

An example of a PSK payload packet is shown.

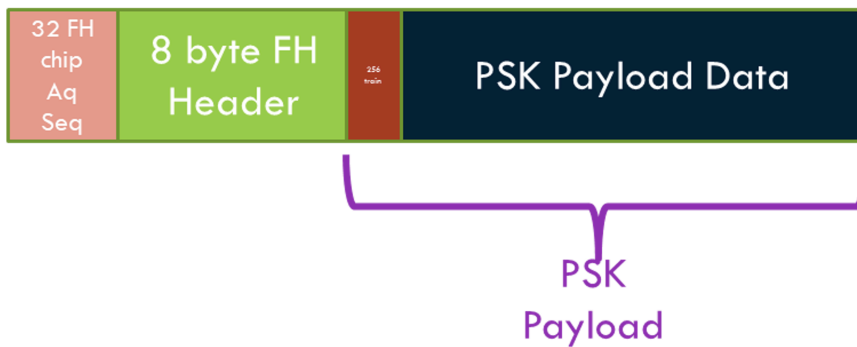


Figure 1.2: PSK Payload Message Structure

The modulation rate of the payload portion of the waveform is configured using the PayloadMode variable. The various modulation schemes are:

- 0 80bps Frequency Hop mode
- 1 5120 bps PSK
- 2 2560 bps PSK
- 3 1280 bps PSK
- 4 640 bps PSK
- 5 10240 bps PSK

The PSK receiver includes user configurable parameters that can be adjusted for optimal reception as a function of the channel. These include the number of taps for the equalizer and the location of the first tap. Under normal operation these parameters are set for typical operation with the number of forward taps (FIR) = 44, the number of backward (IIR) taps=6, and the location of the first tap=16. Note the computational load of the receiver increases with the square of the number of taps and the maximum number of taps (Forward + Backward) should not exceed 70. Also note that additional taps often increase noise and as such more taps does not always mean better performance.

Table 1.1: PSK Parameters

Parameter	Type	Description
PayloadMode	int	Sets the modulation scheme of the transmitter payload <ul style="list-style-type: none"> 0. Frequeuncy Hop FSK 80 Bits/sec 1. PSK 5120 BPS 2. PSK 2560 BPS 3. PSK 1280 BPS 4. PSK 640 BPS 5. PSK 10240 BPS (uncoded)
PSK_FnTaps	int	Sets the number of taps for the FIR portion of the filter (default 44)
PSK_BnTaps	int	Sets the number of taps for the IIR portion of the filter (default 6) <i>Note: PSK_FnTaps+PSK_BnTaps must be less than 70</i>
PSK_StartOffset	int	Sets the location of the first tap in the FIR delay line (default 0) **

1.4.10 Telnet Chat Operation

Lastly, it is possible to open up a chat window between both modems. From a linux prompt on the terminal, type

```
telnet 10.0.0.232 17001
```

this will open a telnet window connected to Modem A (at the 10.0.0.232 address).

Assuming Modem B has been setup with and IP address of 10.0.0.223. Next open another linux prompt on the terminal and type

```
telnet 10.0.0.223 17001
```

This will start another telnet session connected to Modem B (at the 10.0.0.223 address)

Chat operation is a mode of the modem where two modems can communicate keyboard to keyboard in a normal text configuration in a half duplex mode. To enter chat mode the

chat

command is entered at the pshell prompt. It is necessary to enter chat mode at both the receiver and transmitter for chat mode to work.

While in chat mode characters that are entered in the keyboard are grouped into packets and transmitted through the water, received by the receiver and presented to the user.

The start and stop of a packet is determined by 3 factors. The first method is to enter a carriage return after a string of characters. This return signals the end of a string of characters to be sent out of the modem. The second method to signal the end of a string is to timeout. After period of no typing that exceeds the user configurable timeout parameter, the transmitter console will take the user input gathered up until the timeout interval, group them into a packet and send them. The last way to terminate a sequence of characters for transmission is to exceed the user configured number of bytes per packet. For example if the parameter ConsolePacketBytes is set to 32, then input characters are bundled into groups of 32 and sent out automatically.

Table 1.2: Chat Mode Parameters

Parameter	Type	Description
ConsolePacketBytes	int	Sets the number of Sets the number of bytes in the data console after which the console bytes are automatically transmitted
ConsoleTimeoutSec	int	Sets the timeout in milliseconds for the data console, afterwhich any typed bytes are transmitted

To exit chat mode type ctrl-], followed by e for exit. Exit from both the transmitter and receiver to resume normal operation.

1.4.11 Sending a Range Command

Once a successful ping has been achieved, it is instructive to try a range command. The syntax of the range command is as follows:

```
range 4
```

This command instructs the initiating modem (Modem A) to send a range request at the power associated with about 4 watts. You will immediately see the transmitter red LED of Modem A illuminate for about a second. This request should be received by the receiving modem Modem B. Upon successful demodulation of the range request by Modem B, it schedules a transmission back to the receiver of Modem A. This new transmission will illuminate

the red LED of Modem B. When that transmission is complete, Modem A will measure the time required to receive the response to its request, account for turnaround time, and calculate the round trip time. This is mapped to a distance using the speed of sound in water and range is calculated.

Upon successful completion of the whole ranging cycle, a range report will be displayed on the Modem A pshell.

2 OEM Modems

2.1 OEM Modem System

A complete Popoto system consists of the following hardware components:

1. Transducer
2. Analog board
3. Digital board
4. SD card

2.1.1 Analog Board

The Popoto analog board provides signal conditioning to and from the transducer and provides conversion of the analog signals to the digital domain. The signal conditioning of the receiver includes amplification, high pass filtering of the data, and analog to digital conversion. The signal conditioning for transmitter includes digital to analog conversion, and high power transmit amplification.

The analog board also includes a line level analog path to and from SMA connectors for debug purposes. The analog board directly connects to the digital board by way of a 30 position connector at the bottom of the board. A picture of the analog board is shown in Figure ??:

2.1.2 Digital Board

The Popoto digital board provides for all signal processing, interface to analog board, interface digital communication interfaces including:

- RS-232
- RS-422
- Ethernet
- GPIO
- SPI

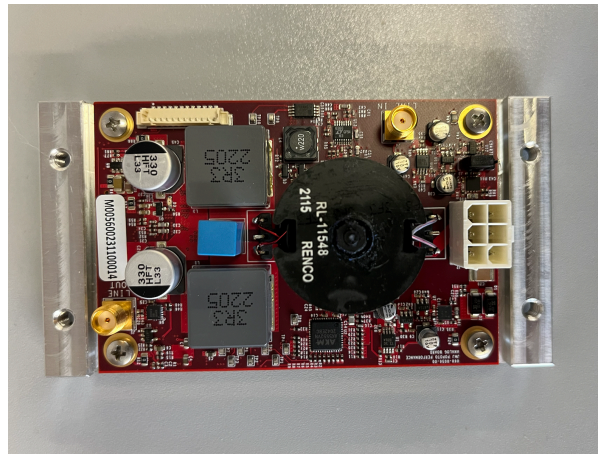


Figure 2.1: The PMM6081 Analog board

- I2C

It also hosts all non-volatile and volatile memory, performs power conditioning, and real-time clock functionality. The heart of this board is an OMAP L138 device made by Texas Instruments. This device includes an ARM 9 host processor which runs Arago Linux, and a TMS320C647x DSP floating point DSP device which performs the computationally intensive signal processing tasks.

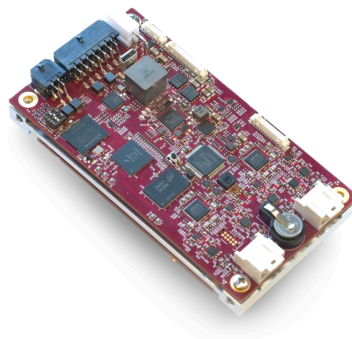


Figure 2.2: The PMM6081 Digital board

2.1.3 micro SD Card

The enclosed Popoto micro SD card has been formatted using Ext4 and includes all of the operating system files, the Popoto application, the DSP application. The SD card includes a directory /captures and provides room for several GB of diagnostic storage if desired.

2.1.4 Heat Sink/Mounting Tray

The Popoto heat sink/mounting tray is used to act as a heat sink for the power amplifier on the underside of the analog board. This heat conductive interface is critical to achieving the full transmit capability of Popoto. The thermal junction between the mounting tray and power amplifier on the analog board requires conductive thermal compound at this interface.

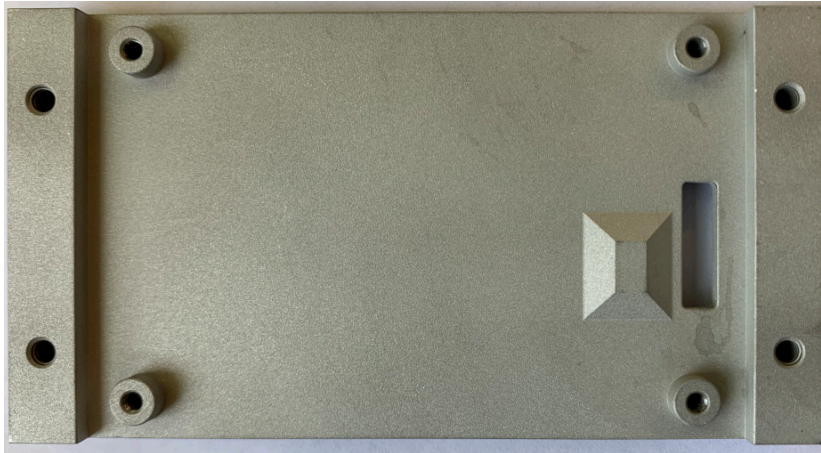


Figure 2.3: The PMM5021 Mounting Tray with M4 mounting screws

3 Lab Evaluation Platform

3.1 Introduction

The Lab Evaluation Setup for the Popoto Modem offers a comprehensive package for developers and researchers to quickly begin working with the Popoto Modem system. This setup is designed to facilitate easy and efficient evaluation of the modem's capabilities in a laboratory environment or for field tests such as pier-side or pool tests.

3.2 What's in the Box

The Lab Evaluation Setup includes the following components:

- An OEM Boardset for the core modem functionality.
- A Modem Interface Board for connections to external devices.
- A Lithium Ion battery to power the setup independently.
- A 3D printed lab evaluation platform that securely holds all components.
- A PMT28 transducer equipped with a cable of sufficient length for versatile testing environments.
- A battery charger for maintaining power supply.
- A USB Cable for connecting the Modem Interface Board to a computer.

3.3 Setup Overview

The Lab Evaluation Setup is designed to provide a rapid, yet inexpensive entry point into the world of underwater communication systems. By combining an OEM Boardset, Modem Interface Board, and PMT28 transducer, users can begin testing and development without the need for extensive setup or additional hardware.

This plug-and-play configuration not only reduces initial costs but also accelerates the process from unboxing to operational testing. The included 3D printed mount fixtures all components securely facilitating tests in a lab environment.

4 DB100 Deckbox

4.1 Description

The DB100 is comprised of a Popoto Modem electronics board set, a Popoto Interface board, and Lithium Ion battery all enclosed in a rugged Pelican 1300 case. All connectors are IP67 splash proof, making for a portable easy to use acoustic communications device. DB100 is designed to be used with an external computation platform such as a laptop or SBC, and can interface to those devices via USB, or via ethernet.



Figure 4.1: The Popoto DB100 Deckbox

4.2 Deckbox Models

The DB100 platform supports all Popoto Modem general-purpose modem boardsets. A low cost PMM3511 based DB100-3 is the entry point to the Popoto Deckbox line, and this line is enhanced by the PMM5021 Based DB100-5. This DB100-5 has 2 further variants beyond the base model, DB100-5-2 Provides an extra hydrophone channel for input, and the DB100-5-SSB incorporates Single sideband voice, with connections for headphones and microphone, as well as a push to talk switch and volume controls. DB100-6 is built around our multichannel PMM6081 and afford additional processing capability for user scripts and signal processing.

4.3 Dunking Transducer

The PMT28-10-DT dunking transducer shipped with the Popoto Modem is cabled with a 10 meter long Polyurethane cable, connected to a 4 Pin JAE connector.

4.4 Front Panel connections

4.4.1 Transducer

To present the information using tables in the 'longtable' format, you will need to first ensure that the 'longtable' package is included in your document preamble. Then, you can replace the 'itemize' sections with 'longtable' environments for each set of pin assignments. This approach is useful for documents that require detailed tables that may span multiple pages. Here's how you can adjust the LaTeX snippet to include these details in tables:

4.4.2 Transducer Connector Specifications on the DB100 Deckbox

The DB100 Deckbox series includes several models, each utilizing different transducer connectors. These connectors are critical for ensuring the correct functionality and compatibility with various transducer and hydrophone systems.

4.4.3 Models DB100-3, DB100-5, and DB100-5-2

For the DB100-3, DB100-5, and DB100-5-2 models, the connector type is a 4-pin JAE: JN1AS04MK1. The pin assignments are detailed in the following table:

Table 4.1: Pin Assignments for DB100-3, DB100-5, and DB100-5-2

Pin Number	Assignment	Notes
1	Transducer Positive	
2	Alternate Channel	Applicable only for DB100-5-2
3	Alternate Channel	Applicable only for DB100-5-2
4	Ground	

4.4.4 Model DB100-6

The DB100-6 model utilizes a 10-pin JN1AS10ML1 connector to accommodate additional hydrophones. The pin assignments for this connector are:

Table 4.2: Pin Assignments for DB100-6

Pin Number	Assignment	Notes
1	Transducer Positive	
2	Hydrophone 1 Positive	
3	Hydrophone 2 Positive	
4	Hydrophone 3 Positive	
5	Hydrophone 4 Positive	
6	Hydrophone 5 Positive	
7	Hydrophone 6 Positive	
8	Hydrophone 7 Positive	
9	Hydrophone 8 Positive	
10	Ground	

4.4.5 USB

The USB port on the front panel of the modem connects to a computer's USB Host port. This connection provides 2 channels of USB-Serial connection. From the factory the first channel is connected to the Pshell, and the second channel is connected to the Popoto Modem's Linux Console. This is generally the fastest way to get started with the Popoto Deckbox.

4.4.6 Ethernet

The eTHERNET port on the front panel is designed to operate with the Stewart XXXXX Ethernet cable. This cable has a locking connector that keeps the connections to the ethernet water tight. Although it is possible to use a standard ethernet cable, it is not recommended, as the IP-rated ethernet connector does not include a provision to latch the ethernet cable in place. Failure to use the proper locking cable will result in unreliable ethernet connections.

4.4.7 Charging

The front panel of the modem features a charging port that utilizes a SwitchCraft 721FMS connector. This connector is designed with a tip-positive pin con-

figuration, ensuring compatibility with specific charging requirements. The Popoto Constant Current Constant Voltage (CCCV) battery charger is designed to connect directly to this port, facilitating efficient charging of the Deckbox battery. This direct connection streamlines the charging process, ensuring that the device is adequately powered for operation.

4.4.8 Headphone and Microphone Jack

For the headphone and microphone interface, the DB100-5-SSB model is equipped with a 3.5mm jack, utilizing a Tensility Intl.: 10-02845 connector. The pinout configuration for this connector is as follows:

Table 4.3: Pinout for Tensility Intl.: 10-02845 Connector

Pin Number	Function
1	Left Headphone
1	Right Headphone
1	Microphone
4	Ground

This connector supports a standard 3.5mm headphone and microphone setup, providing the users with a straightforward means to communicate through the device. The pinout ensures that users can leverage standard audio equipment to interact with the system, enhancing the usability and versatility of the DB100-5-SSB model.

4.5 Single Sideband Voice Feature of the DB100-5-SSB

Please refer to [11](#) for detailed information about the SSB feature of the DB100-5 SSB.

4.6 Getting Started with the DB100

The DB-100 Deckbox offers a seamless way to begin working with the Popoto Modem. It is a self-contained unit that integrates a power source (battery) and all necessary interfaces for connecting to a laptop via USB or Ethernet. For newcomers, the most straightforward method to initiate communication with the deckbox is through a USB connection. This connection enumerates as two serial ports on the laptop, providing access to various functionalities of the Popoto Modem.

4.7 USB Connection

Upon connecting the DB-100 Deckbox to your laptop via USB, it will enumerate as two serial ports. The appearance of these ports varies depending on the operating system:

- **Windows:** The ports are listed as COM ports in the Device Manager.
- **macOS:** Look for devices named `‘/dev/tty.identifier’` in the `‘/dev’` directory.
- **Linux:** Devices appear as `‘/dev/ttyUSBX’` in the `‘/dev’` directory.

The first port provides direct access to the Popoto Pshell, a powerful interface for configuring and controlling the modem.

4.8 Initial Setup

4.8.1 Connecting the Transducer

Begin by connecting the transducer to the transducer connector located on the front panel of the DB-100 Deckbox. Align the arrow on the cable connector with the top mark on the deckbox connector. Gently seat the connector in position before pushing it in until a positive click indicates it is securely connected.

4.8.2 Powering On

Next, connect the USB cable from your PC to the Console port on the DB-100 Deckbox. Power on the unit by pressing the I/O button. Note that, depending on the model, it may take up to 3 seconds for the light around the button to illuminate. Once on, you will be greeted with a `‘Popoto->’` prompt on the first serial port.

4.8.3 Connecting to the Modem

To interact with the modem, use a serial utility such as PuTTY. This will allow you to communicate directly with the modem via the connected serial port.

4.9 Additional Setup for Multiple Units

If you possess two DB-100 Deckboxes, repeat the setup process for the second unit to enable communication between them. For those with a Deckbox and OEM bundle kit, refer to the "Getting Started" section for the initial setup of the additional unit.

If your setup includes a Deckbox and an enclosed unit, see section ?? for instructions on preparing the second unit. Following these steps will allow you to conduct an airstest as detailed in section 1.4.

4.10 Care

To ensure the longevity and optimal performance of the DB100, it is crucial to maintain the cleanliness of its Pelican Case. Given the sensitive electronics housed within, a meticulous approach to cleaning is recommended, focusing on methods that mitigate the risk of water damage. The following guidelines provide a comprehensive approach to safely cleaning the Pelican Case:

1. **Preparation:** Ensure the DB100 is switched off and disconnected from any external power sources.
2. **Dusting:** Utilize a soft, dry cloth or a brush with gentle bristles to dust off the surface of the case. This initial step helps in removing loose dust and debris, reducing the risk of scratches during cleaning.
3. **Spot Cleaning:** For targeted cleaning of spots with stubborn dirt, slightly dampen a soft cloth with water and a mild soap solution, if necessary. It is crucial to ensure the cloth is well-wrung to prevent excess moisture. Gently wipe the affected areas without saturating the case.
4. **Using Spray Cleaners:** If using spray-on cleaners, choose products designed for electronics or those that are mild and non-abrasive. Spray the cleaner onto the cloth rather than directly onto the case to control the amount of moisture applied.
5. **Drying:** After cleaning, thoroughly dry the case with a clean, soft cloth. Confirm the case is completely dry before reconnecting any power sources or using the DB100.
6. **Regular Checks:** Periodically inspect the case for any signs of wear or damage, especially around seals and fasteners. Maintaining the integrity of the case is vital for protecting the DB100 against environmental elements.

Adhering to these cleaning guidelines will help in preserving the condition of your DB100's Pelican Case, ensuring the device remains protected and functional. The emphasis on gentle and dry cleaning methods is paramount to safeguarding the sensitive electronics from moisture-related damages.

5 Enclosed Modems

M2000/M6000/S1000

5.1 Enclosed Modems

5.1.1 Popoto M2000

The **Popoto M2000** modem is the flagship product, featuring a robust design with a machined aluminum housing capable of withstanding depths up to 2000 meters. It exemplifies the brand's commitment to durability and reliability in underwater communication.

5.1.2 Popoto S1000

The **Popoto S1000** emphasizes compactness and efficiency, housed in a machined aluminum casing that withstands up to 1000 meters. With a 20W transmitter, it serves a wide range of underwater communication needs with versatility.

5.1.3 Popoto M6000

Designed for extreme environments, the **Popoto M6000** with its machined titanium shell is capable of withstanding depths of up to 6000 meters. It is the most rugged modem in the lineup, ensuring reliable data transmission in the most challenging conditions.

5.2 Connectors

5.3 M2000/S1000 Connectors

The M2000 and S1000 series modems use SubConn microcircular connectors to provide an electrical interface between the inside of the bottle and the outside. These connectors are typically either MC8BHF for the S Series Modems or MC16BHF connectors. Legacy M2000 Units may be fitted with an MC10BHM 10 pin connector.

Face view (male)

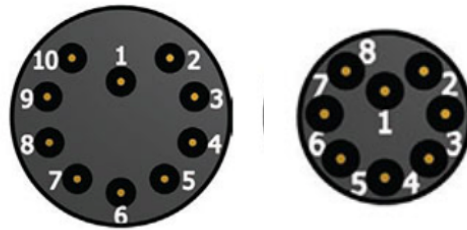


Figure 5.1: Pin Locations: 10 Pin and 8 Pin Connectors as viewed from the face of the male connector.

5.3.1 Connector Part Numbers

Part Number	Description
MCBH16F	16 Pin Bulkhead microcircular female connector
MCBH10M	10 Pin Bulkhead microcircular male connector
MCBH8F	8 Pin Bulkhead microcircular female connector

5.3.2 10 Pin Ethernet Option

Pin Number	Pin Function	Notes
1	Ethernet Tx+	T568A Green White T568B Orange White
2	Ethernet Tx-	T568A Green T568B Orange
3	Ethernet Rx+	T568A Orange & White T568B Green & White
4	Ethernet Rx-	T568A Orange T568B Green
5	RS232 Rx	
6	RS232 Tx	
7	Vin	(+12v to +20v)
8	LED Power	5V out when unit is powered up
9	Gnd	
10	PowerSwitch	Short to ground to power down unit

5.3.3 10 Pin RS-422 Option

Pin Number	Pin Function	Notes
1	RS 422 Rx +(FTDI)	J8-1 RS 422 TX+ (Popoto) Yellow
2	RS 422 Rx -(FTDI)	J8-2 RS433 TX- (Popoto) White
3	RS 422 Tx+(FTDI)	J8-3 RS422 Rx+ (Popoto) Orange
4	RS 422 Tx-(FTDI)	J8-4 RS 422 Rx- (Popoto) Red
5	RS232 Rx	
6	RS232 Tx	
7	Vin	(+12v to +40v)
8	LED Power	3.3V out when unit is powered up
9	Gnd	
10	PowerSwitch	Short to ground to power down unit

5.3.4 16 Pin Universal Option

Pin Number	Pin Function	Notes
1	LED Power	3.3V out when unit is powered up
2	RS 422 Tx+(FTDI)	J8-3 RS422 Rx+ (Popoto) Orange
3	RS 422 Tx-(FTDI)	J8-4 RS 422 Rx- (Popoto) Red
4	RS 422 Rx +(FTDI)	J8-1 RS 422 TX+ (Popoto) Yellow
5	RS 422 Rx -(FTDI)	J8-2 RS433 TX- (Popoto) White
6	PowerSwitch	Short to ground to power down unit
7	Gnd	
8	Ethernet Tx+	T568A Green White T568B Orange White
9	Ethernet Tx-	T568A Green T568B Orange
10	Ethernet Rx+	T568A Orange & White T568B Green & White
11	Ethernet Rx-	T568A Orange T568B Green
12	RS232 Tx	
13	RS232 Rx	
14	PPS Interrupt	Used for PPS in or GPIO
15	Gnd	Ground
16	Vin	(+12v to +40v)

5.3.5 8 Pin Ethernet Option

Pin Number	Color	Pin Function	Notes
1	Red	Ethernet Tx+	T568A Green White T568B Orange White
2	Black	Ethernet Tx-	T568A Green T568B Orange
3	Yellow	Ethernet Rx+	T568A Orange & White T568B Green & White
4	Blue	Ethernet Rx-	T568A Orange T568B Green
5	Orange	Vin	(+12v to +40v)
6	Brown	LED Power	3.3V out when unit is powered up
7	Purple	Gnd	
8	Green	PowerSwitch	Short to ground to power down unit

5.3.6 8 Pin RS-422 Option

Pin Number	Color	Pin Function	Notes
1	Red	RS 422 Rx +(Host)	J8-1 RS 422 TX+ (Popoto) Yellow
2	Black	RS 422 Rx -(Host)	J8-2 RS433 TX- (Popoto) White
3	Yellow	RS 422 Tx+(Host)	J8-3 RS422 Rx+ (Popoto) Orange
4	Blue	RS 422 Tx-(Host)	J8-4 RS 422 Rx- (Popoto) Red
5	Orange	Vin	(+12v to +40v)
6	Brown	LED Power	5V out when unit is powered up
7	Purple	Gnd	
8	Green	PowerSwitch	Short to ground to power down unit

5.3.7 8 Pin RS-232 Option

Pin Number	Color	Pin Function	Notes
1	Red	RS-232 Rx (Host)	J8-12 RS-232 TX (Popoto)
2	Black	RS-232 Tx (Host)	J8-13 RS-232 RX (Popoto)
3	Yellow	GPIO/PPS	J8-14 PPS
4	Blue	No Connect	No Connect
5	Orange	Vin	(+12v to +40v)
6	Brown	LED Power	J8-1 3.3V out when unit is powered up
7	Purple	Gnd	
8	Green	PowerSwitch	J8-6 Short to ground to power down unit

5.4 Lab Cable

Lab cables are a means of connecting Popoto modems to evaluation interfaces, facilitating a straightforward method for testing and exploring modem functionalities in lab settings. These cables are specifically designed to match the connectivity options of different modem models.

5.5 Types of Lab Cables

Depending on the Popoto modem model, there are various lab cables available, each designed to support the interface options of the specific modem.

5.5.1 16-Pin Lab Cable

The 16-pin lab cable is compatible with the **M2000** and **M6000** models, providing RS232, RS422, Ethernet, and control connections for these devices.

5.5.2 8-Pin Lab Cables for S1000

The **S1000** modem, given its compact design, uses a Subconn 8 pin connector. This connector offers tailored connectivity options requiring specific 8-pin lab cables. Users can choose from:

- 8-Pin Ethernet
- 8-Pin RS232
- 8-Pin RS422

It is important to note that the S1000's 8-pin connector does not support all interfaces simultaneously. Thus, the desired interface must be specified at the time of order, and it is important to match the selected configuration with the proper Lab cable.

5.5.3 Extension Cables

For broader deployment applications, the lab cable's reach can be significantly extended using a standard Subconn compatible Male to Female extension cable. This accessory is designed to facilitate flexible deployment configurations, accommodating varying depths and distances as required by the operational environment.

5.5.3.0.1 Availability: The compatible extension cable can be acquired directly from www.popotomodem.com, ensuring that you have access to products designed to work seamlessly with your Popoto modem, providing the following benefits.

- **Operational Flexibility:** By extending the lab cable, modems can be deployed in a wider variety of environments, ensuring optimal placement for data transmission and reception.
- **Customized Deployment:** The availability of extension cables allows for tailored setup configurations, meeting specific depth and distance requirements without compromising the integrity of the connection.
- **Seamless Integration:** Designed specifically for compatibility with Popoto modems, these extension cables provide a reliable and straightforward solution for enhancing your modem's deployment capabilities.

5.6 Power Down and Dummy Plugs

5.6.1 Understanding Power Down Plugs

For ensuring the safety and longevity of Popoto modems during storage or shipment, a special component called the "Power Down Plug" is utilized. This section describes its functionality, usage, and precautions. For uncabled deployment, a dummy plug is included that provides for a water-tight termination of the electrical connections on the bottle.

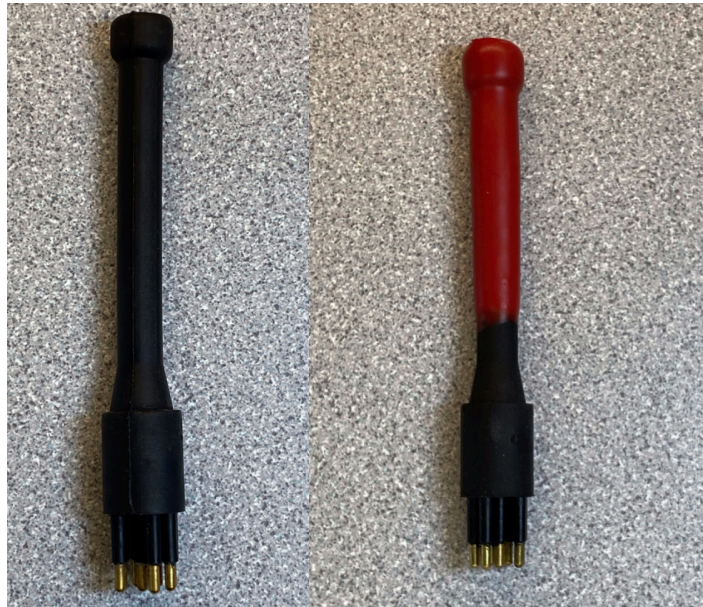
5.6.2 Design and Functionality

Each Popoto modem is equipped with a Subconn Microcircular connector that includes a designated power down pin. This configuration is designed to ensure that the modem can be securely powered down when necessary. The power down process is initiated by connecting the power down pin to the ground pin within the same connector, effectively powering off the unit.

When the power down pin is disconnected, the modem is in the ON state. This method ensures that the modem is maintained in an "on" state, ready for operation when deployed with a dummy plug.

5.6.3 The Power Down Plug

For shipping and storage purposes, a "Power Down Plug" is provided with each modem. This plug is designed to short the appropriate pins, guaranteeing that the unit remains powered off during these periods. It's a critical component for maintaining the modem's safety and ensuring it does not inadvertently power on.



(a) 8 Pin Dummy plug. Note the BLACK rubber whip indicating that this is a dummy plug

(b) 8 Pin Powerdown plug. Note the RED rubber whip indicating that this is NOT a dummy plug

NOTE

DO NOT DEPLOY a modem with a RED whip plug. This plug turns off the modem, and you will not be able to contact it acoustically. All deployments should be with cables, or with BLACK whip dummy plugs.

5.7 Charging

Ensuring your Popoto modem is properly charged is crucial for optimal performance and longevity. This section outlines the correct charging procedure using the designated Popoto Lithium Ion (Li-ion) battery charger, which adheres to the Constant Current Constant Voltage (CCCV) charging protocol.

Each lab cable provided with your enclosed modem features designated battery and ground pins. These are specifically configured for direct connection to the Popoto Li-ion battery charger, ensuring a secure and efficient charging process.

5.7.1 Using the Popoto Li-ion Battery Charger

The Popoto Li-ion battery charger is designed to deliver a CCCV charging cycle, optimal for the specific battery chemistry of the Popoto modem's battery. This charging method is essential for maintaining battery health, preventing overcharging, and ensuring longevity.

5.7.1.1 Indicator LED

The charger is equipped with an indicator LED that communicates the charging status:

- **Red LED:** Indicates that the charger is actively charging the battery.
- **Green LED:** Signals that the charging cycle is complete, and the battery is fully charged.

This visual feedback allows users to easily monitor the charging process and ensures the modem is ready for use when needed.

5.8 Disassembly

While disassembly of the M2000 enclosure may be necessary for maintenance or upgrades, it is important to proceed with caution. The units are vacuum tested before shipping to ensure a tight seal, and disassembly will break this seal, potentially leading to leaks.

5.8.1 Recommendation

It is generally recommended to keep the units sealed whenever possible. If disassembly is performed, it is recommended to conduct a re-vacuum test after reassembly to ensure the integrity of the seal and prevent any possible leaks. Contact Popoto Modem for information on conducting a vacuum test.

5.8.2 M2000/M6000

For a visual guide on the assembly and disassembly process, refer to the official video tutorial available at [M2000 Enclosure Assembly and Disassembly](#) on YouTube.

5.8.3 S1000

For detailed information about disassembling and reassembling the S1000, please contact Popoto Modem.

6 Popoto Transducer (PMT-28)

6.1 Transducer

The transducer for the Popoto Modem is available in three distinct delivery options, each designed to meet specific operational and environmental requirements. This section outlines the configurations and their respective applications.

6.2 Transducer Configurations

6.2.1 Internal Wiring Configuration

The internal wiring option utilizes a shielded, non-waterproof cable. This configuration is ideal for applications where the transducer is housed within a protective enclosure, ensuring the system's integrity against environmental factors.

6.2.2 External Wiring Configuration

For applications requiring direct exposure to the environment, the external wiring configuration employs a durable Polyurethane jacketed cable. This design enhances the transducer's resilience in outdoor or marine settings, offering reliable performance under varying conditions.

6.2.3 Dunking Transducer

The dunking transducer is equipped with a long cable that features a Polyurethane jacket and a strength member. This configuration is suited for applications that involve deploying the transducer into the water from a stationary platform, vessel, or buoy. The inclusion of a strength member ensures the cable's durability and integrity, even under the stress of deployment and retrieval.

All Popoto transducer options consist of a potted piezo ceramic. It is designed to efficiently convert mechanic signal energy to and from electrical analog signals in the 28 KHz region. A picture of the transducer is shown in Figure ??



Figure 6.1: The Popoto 28 Khz Transducer

6.2.4 Transducer Performance

This section provides an overview of the transducer performance for the Popoto modem PMT28, based on comprehensive testing conducted by an independent laboratory. The results offer insights into the transducer's capabilities and efficiency in operational scenarios.

6.3 Performance Curves

The performance of the PMT28 transducer is detailed through a series of curves that illustrate its free field voltage sensitivity, transmit voltage response, and transmit beam pattern. Each of these metrics provides valuable information for assessing the transducer's effectiveness in acoustic communication.

6.3.1 Free Field Voltage Sensitivity

The free field voltage sensitivity curve indicates how effectively the transducer converts acoustic power into electrical signals across a range of frequencies. This metric is crucial for understanding the transducer's ability to detect and process incoming signals.

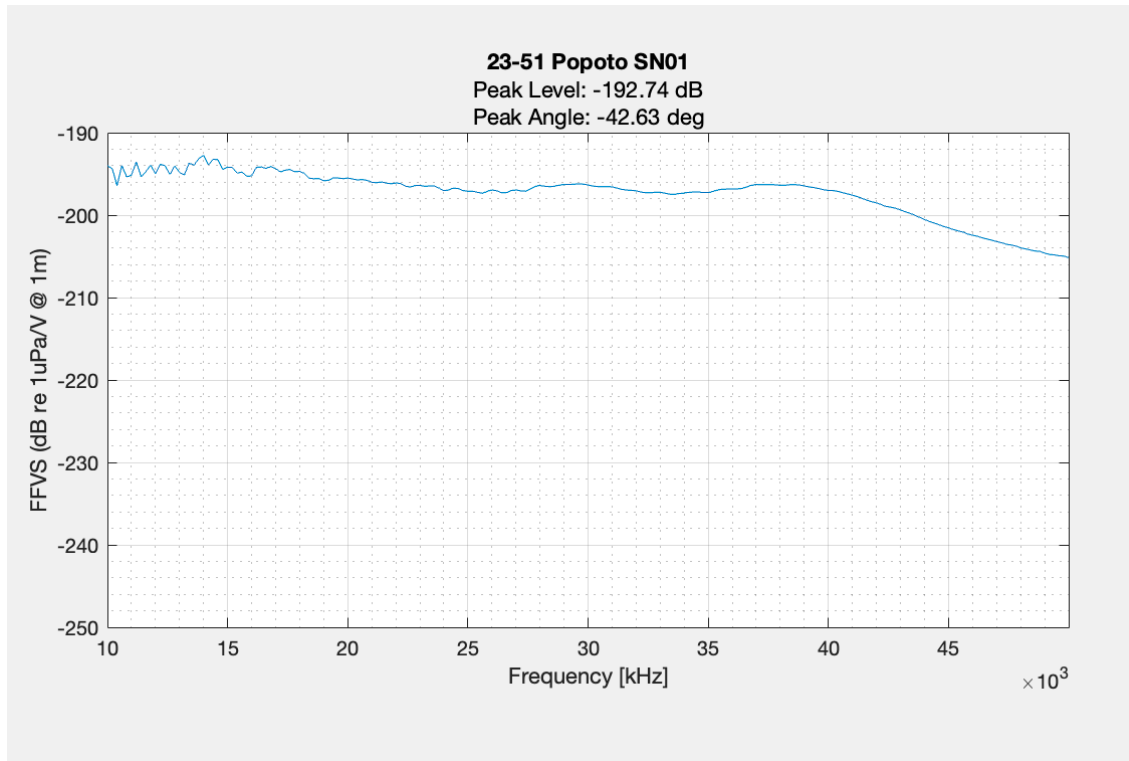


Figure 6.2: The Popoto 28 Khz Transducer Free Field Voltage Sensitivity

6.3.2 Transmit Voltage Response

The transmit voltage response curve demonstrates the transducer’s efficiency in converting electrical signals into acoustic energy. This is essential for evaluating the power output and overall performance of the transducer during transmission.

6.3.3 Transmit Beam Pattern

The transmit beam pattern illustrates the directional characteristics of the transducer’s emitted signal. Understanding the beam pattern is vital for optimizing system setup and ensuring effective communication over the intended area of coverage.

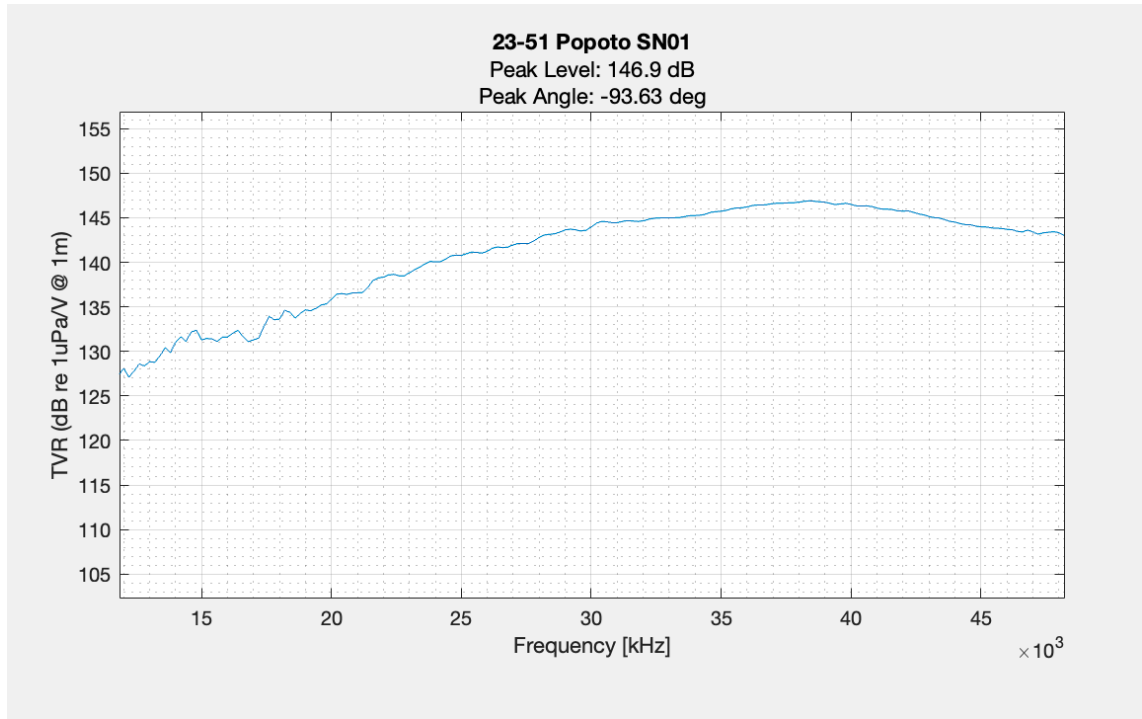


Figure 6.3: The Popoto 28 Khz Transducer Transmit Voltage Response

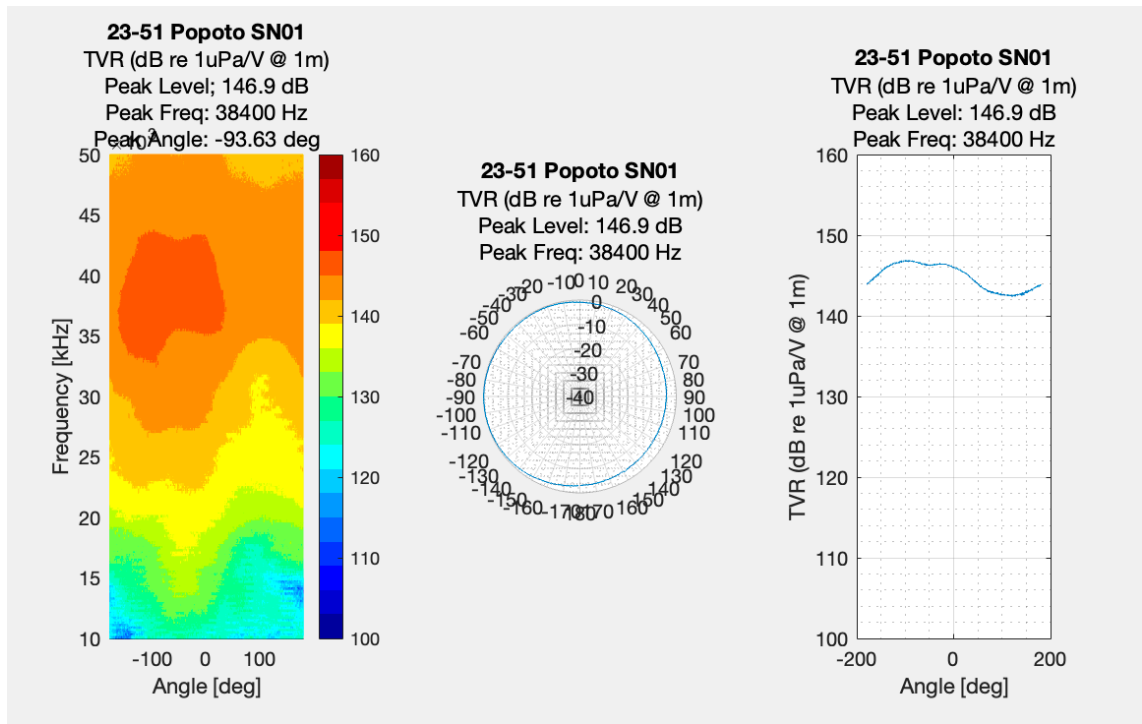


Figure 6.4: The Popoto 28 Khz Transducer Transmit Voltage Response

7 Communicating with Popoto

7.1 Introduction

The Popoto system consists of several components working together to create an acoustic digital communication system. Refer to Figure 7.1. At the lowest level, a Transducer provides the physical interface between the Modem and the water. This transducer is connected to the Analog board which can both drive the transducer as an output, and receive from the transducer as an input. The analog board digitizes input and converts the analog signal in the water to digital data which is sent to the Digital board. The digital board demodulates the data on the DSP, and sends the bitstream to the ARM9 which determines what to do with the data based on the current processing state.

7.2 Socket based JSON

The lingua franca of Popoto is JSON messages over sockets. Although there are many ways and APIs to communicate with Popoto, all of these methods and APIs funnel down to creating or displaying a JSON message to/from a socket.

7.2.1 Highlevel Description of Popoto API Sockets

The IO to all of the embedded Popoto software is accomplished using IP Sockets. Even the analog signal data supports the socket IO. This provides great flexibility for interface, test, software portability, and software test. These sockets also can interface through a thin layer of code to give us the familiar standard interfaces that are used in the field such as RS-422. Sockets are

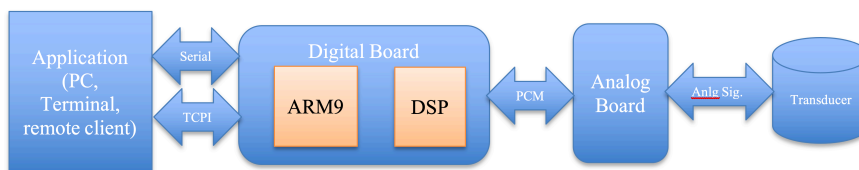


Figure 7.1: Popoto System Overview.

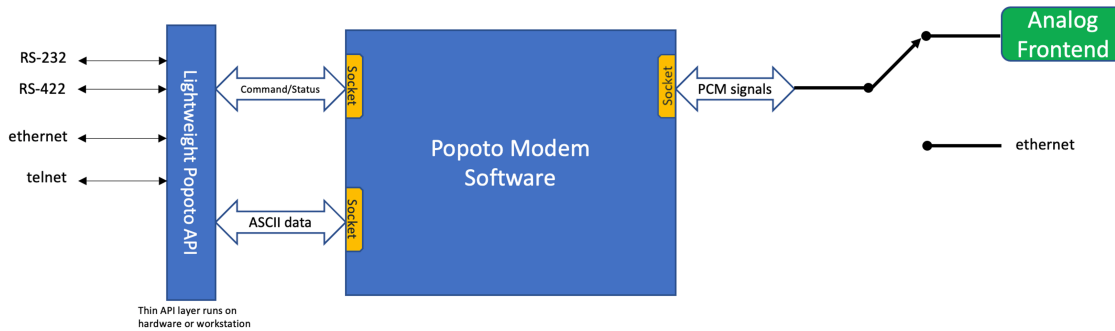


Figure 7.2: Popoto Modem Socket interfaces.

specified by the IP address of the Popoto modem as set by the user. In addition to the IP address the following ports are used.

1. 17000 Command Port
2. 17001 Data Port (Telnet)
3. 17002 PCM Logging Port (Not for typical use)
4. 17003 PCM Output Port (Not for typical use)
5. 17004 PCM Input Port (Not for typical use)

7.2.2 Introduction JSON Messages

JSON stands for JavaScript Object Notation. JSON is a lightweight format for storing and transporting data. JSON is often used when data is sent from a server to a web page. JSON is "self-describing" and easy to understand.

JSON is built on two structures:

- A collection of name/value pairs. In various languages, this is realized as an object, record or struct.
- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

High level languages such as Python typically have JSON parsers available to easily parse JSON messages into variables of that language.

7.2.3 Commands

The basic structure for commanding Popoto happens using a JSON command message. This message consists of two parts, the Command keyword, followed by the Argument keyword. The basic structure of the command is as follows:

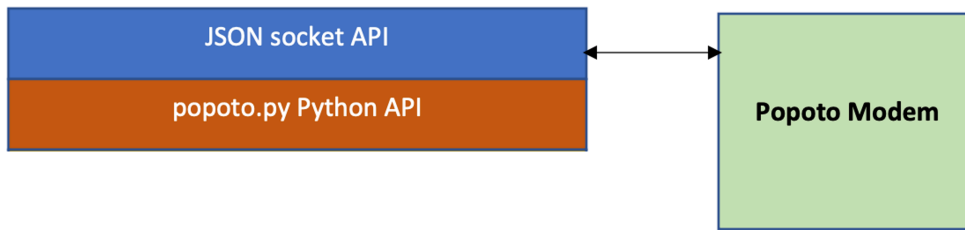


Figure 7.3: JSON API Interfacing to Python.

```
{"Command": "", "Arguments": ""}
```

Example: Get the software version

An example of a simple JSON command is the command to check the software version. This command would be issued as follows:

```
{"Command": "GetVersion", "Arguments": "Unused Arguments"}
```

And would result in the modem responding as follows:

```
{"Info": "Popoto Modem Version 2.7.0"}
```

7.2.4 The Keyword Return Values

Popoto modem returns information to the user using various keyword identifiers. These return keywords are designed to be self-identifying, and can be used for user application parsing.

7.2.5 System Level Variables

Popoto modem contains various internal variables. These variables are mode variables, configuration variables, or contain parameters extracted from the signal.

7.3 Facilitating JSON messages

As mentioned previously, the primary interface to the embedded Popoto algorithm is done over sockets using JSON messages. To make interaction and automated development easiest for a Popoto user, Popoto provides several API and a user shell Pshell. These APIs and shell, constitute a very thin layer that creates and interprets the socket based JSON messages.

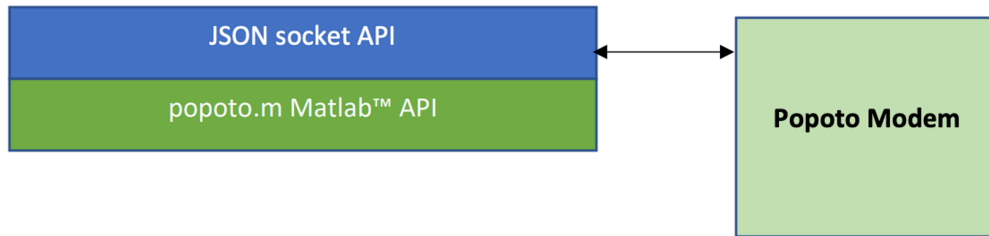


Figure 7.4: Popoto Modem Matlab and JSON API.

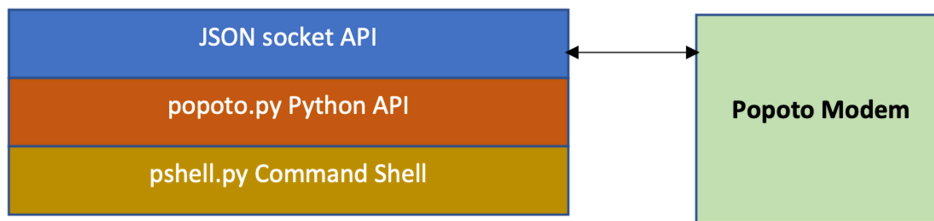


Figure 7.5: Popoto pshell to Popoto.py to JSON.

For example, the popoto.py layer provides python access and methods to create the JSON messages using python. Importing this library gives a user full control over the Popoto modem in the python language.

Much in the same way, a Matlab™ based API has been generated. The structure of how it interacts with Popoto is the same. Finally, a command shell called Pshell has been written in python, using the popoto.py API and the cmd command interpreter. This shell allows users to interact with the modem at a user level, typically through a serial connection. The pshell is default way for a user to interact with Popoto modems. The structure of this connection is a follows:

8 System Connections

The Popoto modem has 4 primary interfaces for an external CPU or computer to connect to:

1. RS422 4 Wire serial
2. RS-232 Uart
3. 10/100 BaseT networking
4. TTL (3.3V) UART

Each of these connections has properties that make it attractive in different situations.

8.1 RS-422 4 wire serial

8.1.1 Reasons to use it

Good for long distance connections, up to 1200 meters. Simple serial interface. Robust to noise and interference.

8.1.2 Reasons to avoid it

Remote unit needs drivers. Only good for up to 115200 bits per second which is not adequate for PCM Streaming.

8.2 RS-232 Uart

8.2.1 Reasons to use it

The RS-232 uart is a 3 wire serial interface. It consists of 3 signals, Transmit data, Receive Data, and ground. The signal lines run at +/-15 Volts This interface is particularly attractive if the user is interfacing the modem to a local device, such as a micro controller on a UAV. All that is required is a tx, rx and gnd signal. For PC or laptop lab use, the pinout for this connector is a 5 pin 10mil header configured exactly as the standard FTDI USB cables, which makes for a simple USB to serial interface available off the shelf.

8.2.2 Reasons to avoid it

This interface is only good for very short distances, such as within the same enclosure. The bandwidth for this interface is limited to 115200 Bits per second which is not adequate for PCM streaming.

8.3 10/100BaseT

8.3.1 Reasons to Use it

The 10-100BaseT Ethernet networking provides the highest speed and most flexible connection to the Popoto system. Using TCP sockets over the ethernet provides upto 100MBits/S of full- duplex throughput to the Popoto from a remote computer located up to 100 meters away. This bandwidth can be used for real-time PCM capture, or rapidly updating software. Additionally, the flexibility of the TCP sockets allows for 3

8.3.2 Reasons to avoid it

The additional speed and flexibility of the ethernet comes at a cost of 250 milliwatts. In addition, the range of the ethernet is limited to 100 meters.

8.4 TTL (3.3V) UART

8.4.1 Reasons to use it

The TTL (3.3V) uart is a 3 wire serial interface. It consists of 3 signals, Transmit data, Receive Data, and ground. The signal lines run at 3.3Volts This interface is particularly attractive if the user is interfacing the modem to a local device, such as a micro controller on a UAV. All that is required is a tx, rx and gnd signal. For PC or laptop lab use, the pinout for this connector is a 5 pin 10mil header configured exactly as the standard FTDI USB cables, which makes for a simple USB to serial interface available off the shelf.

8.4.2 Reasons to avoid it

This interface is only good for very short distances, such as within the same enclosure. The bandwidth for this interface is limited to 115200 Bits per second which is not adequate for PCM streaming.

8.5 Modes of operations

The flexibility of the Popoto system provides for several use-cases. Each of these use-cases applies to a different product scenario, so it is important when deciding which to employ, that the requirements of the end product are carefully considered.

8.5.1 Local pshell

The pshell is a python program that connects with the Popoto application and provides a shell interface to the modem and its command, status and data interfaces. This shell provides simple commands such as send ranging, or setTxPower level so that either under human or computer control the modem can be utilized. In this use-case, the interface to the modem can be any one of:

- Serial RS-232
- 4 Wire RS-422
- Ethernet over SSH
- Ethernet over Telnet

The pshell program runs co-resident with the popoto_app on the OMAP's ARM core processor. Figure 8.1 shows local pshell processing.

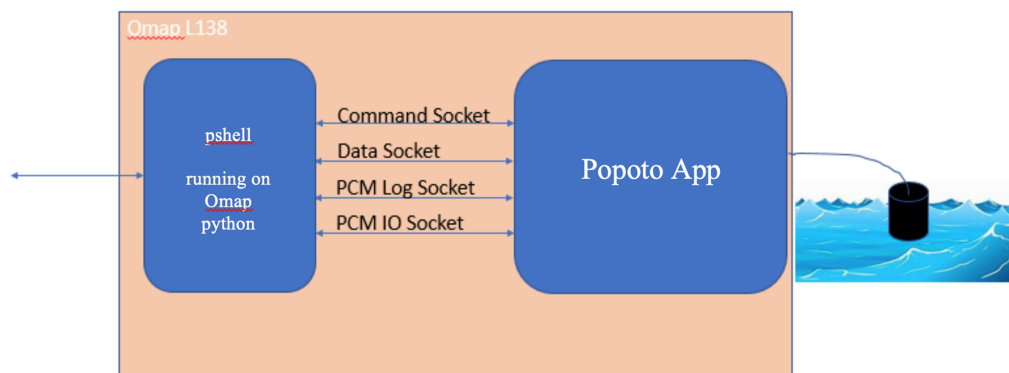


Figure 8.1: Local Pshell operation.

8.5.2 Remote pshell

The remote pshell operates in the same way as local pshell, however the pshell python program runs on a remote processor, and the connection to the popoto_app is over TCP Sockets and networks as shown in figure 8.2. Using a remote pshell is advantageous for streaming PCM directly to the PC's harddrive. Additionally the remote pshell is a good choice for running regression tests, as the regression suites can live on the remote pc, which can also log results.

8.5.3 Matlab™

Matlab™mode is very similar to remote-pshell mode, except that the connection to the popoto_app does not use a Python program, rather it connects

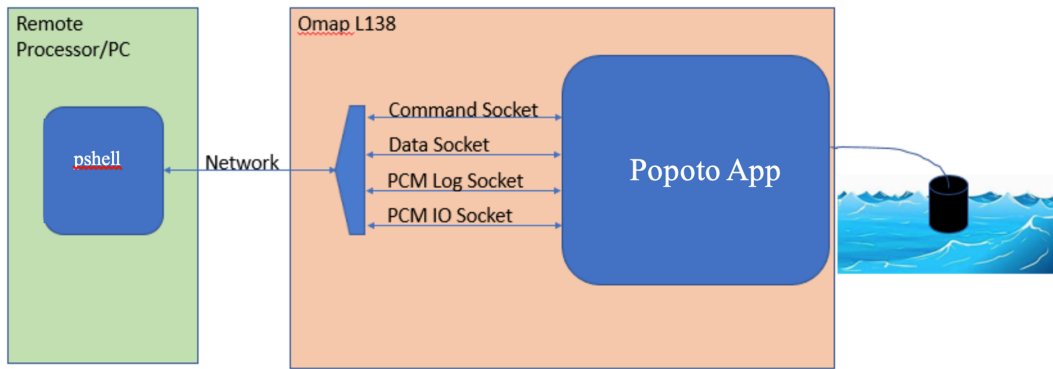


Figure 8.2: Connecting to Popoto using a remote pshell

using Matlab™. Matlab™ is an excellent choice for running lab tests as it is a powerful language that is easy to use. Given Matlab™'s expense, and need for a full PC to run, it is not likely to be deployed in a customer's end product.

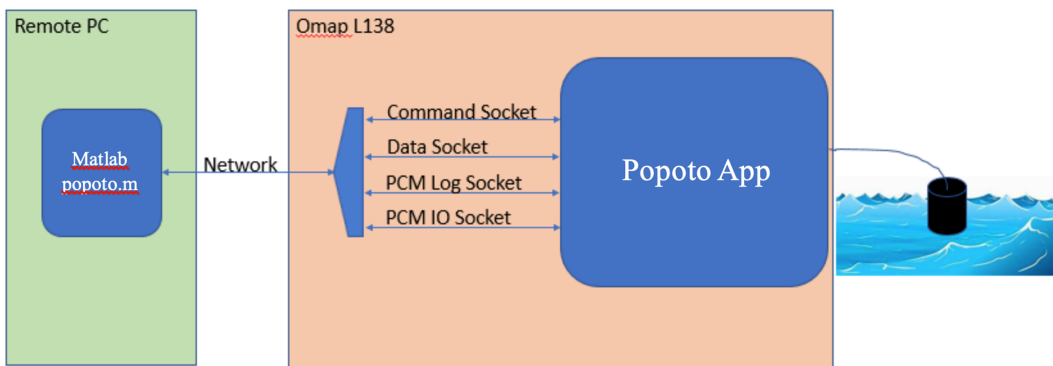


Figure 8.3: Connecting to Popoto Over Matlab™.

8.5.4 Custom interfaces

The Popoto system uses standard sockets for communications, so it is entirely possible for a customer to generate a custom interface written in the language of his choice. Figure 8.4 shows an example of a custom Popoto application. Please see the Popoto.py and Popoto.m files for ideas on how to implement such an interface.

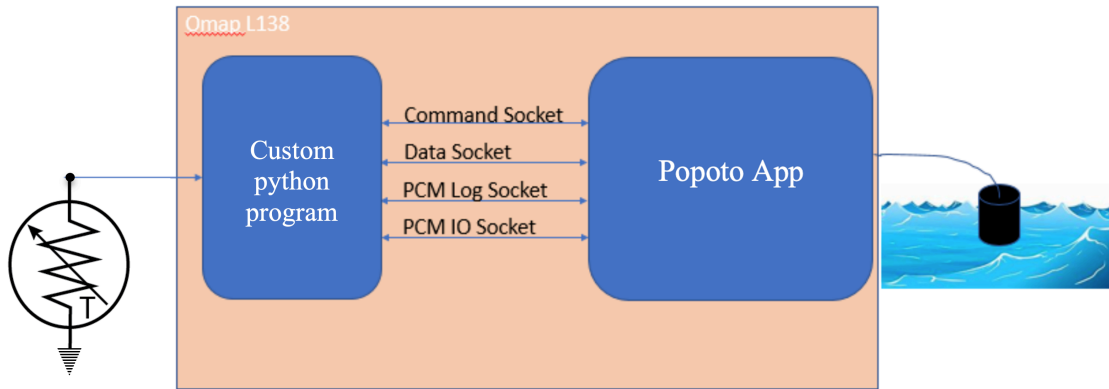


Figure 8.4: Popoto Modem implementing a custom application. In this picture, a Popoto modem is configured to measure a temperature sensor, and report its measurements via the acoustic channel.

8.6 Sleep and Power Down

Popoto has 2 different sleep modes for low-power operation, POWERDOWN and DEEPSLEEP. In Powerdown mode, nearly all of the voltage supplies on the board are shut down, resulting in a very low-power sleep. It takes about 20 seconds to power up and out of POWERDOWN mode. In DEEPSLEEP mode, the processor is put into hibernation, and all RAM is kept powered and refreshed. This mode consumes more power, but wakes up quite quickly (<1Sec). Each mode can be entered via an API or pshell command. The pshell exposes 2 commands powerdown, and deepsleep. Refer to the Popoto API reference for information about API access to the powerdown states. In either powerdown mode, a dedicated wake-up processor monitors the acoustic signals in the water looking for a wakeup signal. The wakeup signal is based on the standard modem message header acquisition pattern. To wake a modem, simply send any message into the water, and the acquisition signal will wake the modem. This transmitted message will be lost, but it serves simply to wake the unit.

9 Pshell

The pshell is a python command line shell utilizes commands defined in Popoto.py to provide a python scriptable command shell containing all of the most useful commands from Popoto.py. In addition the command shell provides for help and tab completion for ease of use. Responses from commands are echoed to the command shell along with asynchronous alerts from Popoto.

9.1 Modes of operation

There are two fundamental modes of operation of the pshell, it can be run on the user PC under the PC's local python or it can be run on the python that exists on Popoto OMAP platform. Because communication from pshell to the Popoto is done through IP sockets, this gives the flexibility of running pshell locally on the target or remotely on any PC on the network.

9.2 Requirements for running

python 2.7 (it is already installed on the Popoto hardware) CMD command shell installed (it is already installed on the Popoto hardware) CMD2 command shell installed (this gives some added features)

9.3 Invoking pshell

As delivered, the Popoto will invoke the pshell automatically and present a command prompt to the user on the RS-422 port.

9.3.1 The pshell.init file

The pshell.init file is located in the root directory /. This file is a collection of pshell commands that get executed on power up of Popoto. This file is intended for the user to customize this file and set bootable parameters such as localID, carrier frequency etc. The syntax is normal pshell syntax where line comment character # (first position) and whitespace are ignored.

9.3.2 Invoking pshell from a linux prompt

Although pshell runs automatically at boot. It is possible to terminate the local running pshell process and run the pshell from any python with an IP connection to Popoto. From the linux command prompt

```
python pshell
```

This will start the up the pshell and you are ready to being typing commands.

9.4 Invoking commands

9.4.1 Help

To gain a complete list of commands at any time simply type the command help. A full list of commands will be displayed. To get help on any of those commands, enter help <command> at the Popoto prompt.

9.4.2 Tab Completion

The pshell supports tab completion. Tab completion will also show a list of various options for a particular command.

9.4.3 Commands

This section presents a list of the currently implemented commands. A brief description is presented along with typical invocations.

9.5 Extending the pshell

One of the best parts of pshell is that it is easy to extend with simple python. For example if you want to make a command that does five ranges spaced by 30 seconds, it is as simple as adding these lines:

```
def do_nranges(self,line):
    for x in range(1,5):
        self.dol.range(.1)
        time.sleep(30.)
```

Note the command name in the pshell would be nranges. With the pshell, you have the power of the python language to create complex commands or specific syntaxes, mappings, command checking etc very quickly and efficiently.

9.6 Scheduling Transmissions

The Popoto Modem API offers a feature allowing users to schedule the transmission of a data packet at a specific time, utilizing the PCM sample count for precision. This document guides you through using this feature.

Understanding PCM Sample Time

Time within the Popoto Modem is measured in PCM samples, with a frequency of 102,400 samples per second. The `PCMSampleCounter` variable holds a 32-bit count of these samples, offering a high-resolution time base for scheduling transmissions.

Querying the PCMSampleCounter

To retrieve the current PCM sample count, use one of the following methods:

- Send the command `get PCMSampleCounter` to the modem.
- From the pshell, type `PCMSampleCounter` and press Enter.

In versions 3.2.6.3, 3.2.8, or later, the `PCMSampleCounter` at the start of each received packet is reported in a JSON message, e.g., `{"PCMSampleCount":4446530}`.

Scheduling a Transmission

To schedule the transmission of a packet, use the `transmitJSON` command with a specified departure time in PCM sample count. Here is the syntax for scheduling a transmission:

```
pshell> transmitJSON {"Payload":{"Data":"Hello"}, "DepartureTime":30000000}
```

This command schedules the packet containing the message "Hello" to be transmitted at the PCM sample count of 30,000,000.

Transmission Status

After scheduling a transmission, the modem reports the status through a series of JSON messages, indicating the scheduled departure time, the actual PCM sample count at transmission, and various modem settings adjustments. For example:

```
1 {"DepartureTime":150000000}
2 {"PCMSampleCount":138576000}
3 {"Info":"Value Set MODEM_Enable=1"}
4 {"Info":"Value Set MODEM_Enable=1"}
5 {"Alert":"TxComplete"}
6 {"Info":"Value Set MODEM_Enable=1"}
7 {"Info":"Value Set RxEnable=1"}
```

Handling Early or Past Scheduled Transmissions

It is important to note that scheduling a transmission to occur too soon (considering it takes approximately 300ms to power on the transmit Power Amplifier) or in the past will result in the transmission happening as soon as the hardware is ready. If a requested departure time is too early or has already passed, you will receive status messages indicating this, alongside the actual transmission details:

```
1 {"Info":"Requested Departure Time Too Early or in the Past. "}
2 {"DepartureTime":130555520}
3 {"PCMSampleCount":130483840}
4 {"Info":"Value Set MODEM_Enable=1"}
5 {"Info":"Value Set MODEM_Enable=1"}
6 {"Alert":"TxComplete"}
7 {"Info":"Value Set MODEM_Enable=1"}
8 {"Info":"Value Set RxEnable=1"}
```

9.6.1 Streaming Mode Operation for Popoto Modem

9.6.1.1 Introduction

9.6.1.1.1 Purpose This document explains the configuration and operational benefits of Streaming mode on the Popoto modem. This mode is designed to handle larger data packets efficiently, which is critical for applications requiring robust and continuous data transmission.

9.6.1.1.2 Scope The guide outlines the necessary settings for Streaming mode and quantifies the benefits in terms of transmission time savings.

9.6.1.2 Configuration

9.6.1.2.1 Enabling Streaming Mode To activate Streaming mode, set the following variables:

- **ConsolePacketBytes:** Set to 256. This configuration specifies the packet size that the modem processes.
- **StreamingTxLen:** Set to 2048. This setting allows the modem to consolidate transmitted data into super packets or frames, reducing the frequency of required headers.

9.6.1.3 Operation

9.6.1.3.1 Sending Data In Streaming mode, transmitJSON messages of 256 bytes are sent continuously without the need for waiting for transmit complete messages. This setup significantly streamlines the data flow to the modem.

9.6.1.3.2 Data Handling by the Modem The modem software queues the incoming 256-byte messages and automatically combines them into super frames of 2048 bytes. This process minimizes communication overhead and increases data throughput.

9.6.1.4 Efficiency Gains

9.6.1.4.1 Header Duration and Transmission Savings Each header in Streaming mode consists of 176 chips (32 + 128 + 16 chips), with each chip lasting 6.25 milliseconds. The duration of one header is therefore 1100 milliseconds. Unlike the standard mode, where each packet would require a new header, Streaming mode only necessitates a header for each super packet (2048 bytes).

- **Header Duration:** 1100 ms per super packet.
- **Savings:** By transmitting seven fewer headers for each super packet, Streaming mode saves 7700 milliseconds per super packet compared to non-streaming mode.

9.6.1.4.2 Mathematical Expression of Savings The time saved per super packet in Streaming mode is calculated as:

$$Savings = HeaderDuration \times 7 = 7700ms$$

10 Popoto Modulation Data Rates and Payload Modes

Popoto modems support a family of payload transmission modes designed to operate across a wide range of underwater acoustic channel conditions. All coherent modes share the same occupied bandwidth and symbol rate and employ a common 1/2-rate convolutional Forward Error Correction (FEC) code. Robustness is controlled through repetition coding: lower bitrates correspond to higher redundancy and increased tolerance to low-SNR, multipath, or Doppler-degraded environments.

An additional uncoded QPSK mode is provided for advanced users who wish to apply external error-correction schemes. A frequency-hopped FSK mode provides extreme robustness in the lowest-SNR environments.

This chapter describes each payload mode, its coding structure, and measured SNR operating thresholds.

10.1 PayloadMode Enumeration

The modem supports the following PayloadMode values:

```
FH_80bps = 0,  
QPSK_5120bps,  
QPSK_2560bps,  
QPSK_1920bps,  
QPSK_1280bps,  
QPSK_10240bps,
```

```
BPSK_2560bps,  
BPSK_1280bps,  
BPSK_960bps,  
BPSK_640bps,
```

These modes fall into three families:

- FH-FSK (80 bps)
- BPSK family (640–2560 bps)
- QPSK family (1280–10240 bps)

10.2 Coding Structure

10.2.1 Coherent Modes (BPSK & QPSK)

All coherent payload modes use:

- A common 1/2-rate convolutional FEC (K=7)
- Identical symbol rate and bandwidth
- Repetition coding to adjust robustness

Changing the payload mode does **not** change Doppler tolerance or multipath behavior. The only variable is the *processing gain* introduced by additional redundancy.

10.2.2 Uncoded Mode (QPSK_10240bps)

QPSK_10240bps bypasses FEC and repetition entirely. It doubles the information rate of QPSK_5120bps but requires significantly higher SNR. This mode is intended for users implementing custom external FEC such as LDPC, Turbo, or Fountain codes.

10.2.3 FH-FSK Mode

FH_80bps uses a frequency-hopped narrowband FSK waveform selected for extremely low-SNR or highly variable acoustic environments. It provides the highest robustness and is useful for long-range or minimal-SNR command and control links.

10.3 Recommended Modulation Use

For all payload rates at **2560 bps and below**, Popoto recommends the use of **BPSK**. BPSK offers:

- Lower SNR requirement than QPSK (typically 2–3 dB advantage)
- Greater robustness under multipath and platform motion
- Wider operational envelope in low-SNR coastal environments

The QPSK modes at 2560 bps and below are retained for **backward compatibility only**. They remain supported for legacy systems but are not recommended for new applications unless required for interoperability.

Table 10.1: PayloadMode Summary

Mode	Mod	Rate (bps)	FEC	Repetition	Recommended	Notes
FH_80bps	FH-FSK	80	inherent	n/a	Yes	Extreme low-SNR
BPSK_640bps	BPSK	640	1/2	4x	Yes	Most robust cohe
BPSK_960bps	BPSK	960	1/2	2.67x	Yes	Long-range, low-
BPSK_1280bps	BPSK	1280	1/2	2x	Yes	Medium/long ran
BPSK_2560bps	BPSK	2560	1/2	1x	Yes	Baseline BPSK
QPSK_1280bps	QPSK	1280	1/2	4x	No	Backward compa
QPSK_1920bps	QPSK	1920	1/2	2.67x	No	Backward compa
QPSK_2560bps	QPSK	2560	1/2	2x	No	Backward compa
QPSK_5120bps	QPSK	5120	1/2	1x	Yes	High-rate coded
QPSK_10240bps	QPSK	10240	None	None	Conditional	Requires externa

10.4 Payload Mode Summary

10.5 Measured SNR Waterfall Thresholds

Table 10.2 summarizes the approximate SNR breakpoints observed in representative coastal environments where packet error rate increases sharply.

Table 10.2: Measured SNR Waterfall Thresholds

Mode	SNR (dB)	Observed Behavior
FH_80bps	-5	Maintains lock at very low SNR
BPSK_640bps	2	Strongest coherent mode
BPSK_960bps	2.5-3	High redundancy
BPSK_1280bps	≈ 3	Medium/long range
BPSK_2560bps	3	Baseline coded BPSK
QPSK_1280bps	≈ 4	Backward compatible only
QPSK_1920bps	4.5-5	Backward compatible only
QPSK_2560bps	5-6	Backward compatible only
QPSK_5120bps	5	Recommended coded QPSK
QPSK_10240bps	10	Uncoded; short-range only

10.6 Mode Selection Summary

- **Highest robustness:** FH_80bps, BPSK_640bps
- **Low-SNR or long-range:** BPSK_960bps, BPSK_1280bps
- **General operation at moderate SNR:** BPSK_2560bps
- **High-rate short-range bursts:** QPSK_5120bps

- **External FEC research modes:** QPSK_10240bps
- **Legacy interoperability:** QPSK_2560bps and below

11 Single Side Band Voice Operation

12 Single Side Band Voice Operation

12.1 Overview

Utilizing Popoto's single sideband transmitter (SSB) and receiver allow for half duplex voice communication through the water. The SSB signal is inherently an analog signal being through the ocean at a carrier frequency. As an analog signal, this means that the reception of the analog waveform includes the analog impairments of the channel. So if the channel is noisy, the receiver will hear the noise. If the channel has echo, the resulting speech will include echo. If there is no noise and no echo, and analog levels are set properly, there will be no distortion of speech aside of the normal band limiting associated with telecom speech. To utilize the SSB functionality of Popoto, it is necessary to ensure that the voice path electronics are powered up. This is done by ensuring jumpers J1 and J5 are populated.

12.2 SSB Transmitter

The transmitter consists of a single sideband modulator which receives speech from the microphone input J3 and modulates it up to carrier for transmission out of the transducer and through the water. There are 3 ways to place the SSB transmitter in transmit mode.

1. A Popoto ssbtx command
2. A hardware PTT signal
3. Using the properly adjusted VOX

12.2.1 The ssbtx command

Issuing the ssbtx command places Popoto in transmit mode. This can be clearly seen by the transmit LED glowing red on the analog board. Once in transmit mode, audio that is input on SMA J3 will be modulated, shifted up to carrier, power amplified, and delivered to the transducer.

12.2.2 Adjustment of transmit power

Proper adjustment of the transmit power is critical for good operation of the SSB transmitter. Setting this power properly is a function of 2 variables

1. Microphone sensitivity
2. Desired transmit power

Both of these variables is are controlled by the SSB_Txpower variable. This variable should be set such that the desired PEP power is achieved while speaking at a normal level in the microphone.

12.2.3 Peak Envelope Power

The proper adjustment of power for voice operation revolves around properly setting the Peak Envelope Power. PEP is the value of power that is output by the transmitter when the speech is at peaks in its overall envelope. Typically average power of speech is between 10%-20% of the peak envelope power. These adjustments should be made while the transducer is in water. Also these setting can be approximated by careful monitoring of the input power in these peak regions and setting the SSB_Txpower constant appropriately. Choosing an appropriate PEP level is a function of the distance that one wishes to transmit, the SNR of the channel, along with the reflectivity of the channel. These settings can be experimentally derived in the water and presets can be made in the pshell for optimum speech quality.

12.2.4 PTT keying of the transmitter

The Popoto hardware presently includes two GPIOs that are used for PTT and also headset volume control. The truth table shown below illustrates the various modes associated with the GPIOs. When the two GPIOs are zero, the transmitter is keyed, when they are 1,1 the receiver operates, and the other two states will raise or lower the headset volume by 1 dB per click.

Table 12.1: SSB Control Bits

Gpio8[6]	Gpio7[14]	State
0	0	PTT Depressed (Transmit Mode)
0	1	Headphone Volume UP
1	0	Headphone Volume DOWN
1	1	Receive Mode **

In order to work with the PTT and VOL UP VOL DN the J7 Pin 3 and J7 Pin 9 Must be pulled up to 5V through their own individual 4.7K Resistors.

Table 12.2: SSB Pinout

Function	GPIO Number	Processor Signal	J7 Pin
VOL UP/PTT	GPIO8[6]	SPI_SOMI	J7_3
VOL DN/PTT	GPIO7[14]	EXP1	J7_9

12.2.5 Transmitter Vox

The SSB transmitter can be switched on using the speech signal itself. To utilize this feature, the SSB_Vxmode should be set to 1. Next the SSB_Vxlevel should be increase from zero slowly while speaking to arrive at the trigger point for the VOX. Proper setting of this level will ensure that constant level audio background will not trigger the transmitter, but onsets of speech will trigger the transmitter. Note that once the transmitter is keyed, the transmitter remains on for a period of 2 seconds.

12.3 SSB Receiver

Voice mode reception is enabled by issuing the `ssb` command from the `pshell`. At this command the modem will transition from data modem mode to single side band receiver. Demodulated audio will be present on the SMA connector J4. The audio level present on J4 is controllable by setting the `SSB_Volume` parameter to the user desired level.

12.3.1 Squelch

Additionally, the receiver incorporates a squelch for eliminating background noise between segments of received speech. To utilize the squelch it is important to set the `SSB_SqLevel` parameter in the `pshell`. An `SSB_SqLevel` of zero reflects no squelch and the receiver will be continuously in the receive state with demodulated audio being presented to the headphones. The user can gradually increase the squelch level until the interspeech segments are muted.

12.3.2 Noise Reduction and AGC

The SSB Receiver incorporates an advanced Noise reduction and AGC which are enabled using the `SSB_NREnable` flag as detailed in Table 12.3. This module monitors the background noise using spectral analysis to provide an adaptive noise reduction and speech quality improvement. Enabling this mode also enables the receiver automatic gain control. Using the noise reduction algorithm's internal metrics, the automatic gain control is able to determine when speech signals are present in the received signal. During these times, the receiver gain is adjusted to provide speech at an audible signal level.

12.4 Return to data mode

To return to data mode simply enter `datamode` at the pshell prompt.

12.5 SSB Controllable parameters

The table below show all of the settable/gettable parameters available through the pshell for the purpose of controlling SSB operation.

Table 12.3: SSB Control Bits

Parameter	Type	Description
SSB_Volume	float	Sets volume level of headset
SSB_Txpower	float	Sets microphone gain; for SSB, this controls the Tx power
SSB_VxMode	int	Normal PTT set to 0; VOX mode PTT set to 1
SSB_VxLevel	float	Sets the trigger level for PTT VOX
SSB_SqLevel	float	Sets the background noise level for squelch trigger (0.-always on)
SSB_NREnable	int	Enable or disable the Noise reduction/AGC algorithm 1=On 0=Off

13 Janus Operation

13.1 Janus Overview

Janus is a standardized physical and data link layer for acoustic underwater communications designed and implemented by NATO's Centre for Maritime Research and Experimentation (CMRE). The standard is described in the NATO [ANEP-87](#) specification. Further details about Janus are available at the [wiki page maintained by CMRE](#).

13.2 Janus Bitstream

One of the defining features of Janus is its configurable payload scheme. Each message begins with an acquisition pattern, followed by modulated bits. These definition of these bitfields is variable based on the message content. The baseline bit allocation table for the Janus message is shown in Figure 13.1. In this figure, you will notice that bits 31-56 are referred to as the application data block. The definition of this block of bits is dependent on the Class UserID and Application Type bitfields found earlier in the message.

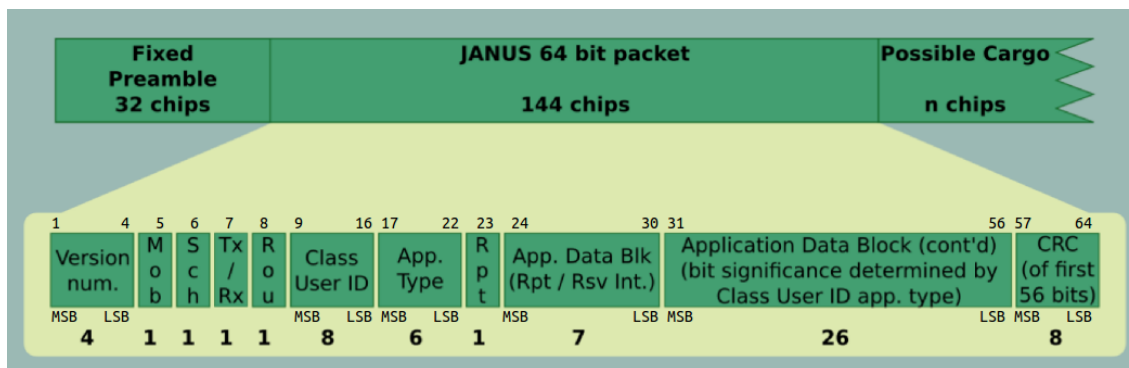


Figure 13.1: Janus Baseline Bit Allocation. (from the Januswiki.com webiste)

13.3 Popoto Modem Implementation of Janus

The Popoto Janus modem fully implements all of the ratified Janus Class ID's, which currently include Class ID's 0, 2, and 16. Within the Popoto modem, each of the fields exported in the Class definition are represented by a field in a JSON encoded message. In this way, a JSON encoded message can be translated directly into a Janus message class. What follows are examples of each of the supported message types along with a link to the Class description on the JanusWiki.com website.

13.3.1 Class User Id: 000 Emergency

13.3.1.1 ApplicationType 1: Position Message

Refer to the wiki page [Emergency Position Message Format](#) for details about the fields associated with the Emergency position message. The example below shows how to construct a representative ClassID 000 ApplicationType 1 Emergency Position message for transmission by the Popoto Modem. This JSON structure would be encoded on a single line, and passed to the transmitJSON api of the pshell, or other Popoto API (C++, Python, Matlab).

```
1 {  
2   "ClassUserID": 0,  
3   "ApplicationType": 1,  
4   "Nationality": "AB",  
5   "Latitude": "90.000000",  
6   "Longitude": "0.000000",  
7   "Depth": "8190",  
8   "Speed": "0.000000",  
9   "Heading": "180",  
10  "MobilityFlag": "1",  
11  "ForwardingCapability": "1",  
12  "TxRxFlag": "0",  
13  "ScheduleFlag": "1",  
14  "Schedule": "10"  
15 }
```

13.3.1.2 ApplicationType 2: Status Message

Refer to the wiki page [Emergency Status Message Format](#) for details about the fields associated with the Status message. The example below shows how to construct a representative ClassID 000 ApplicationType 2 Status message for transmission by the Popoto Modem. This JSON structure would be encoded on a single line, and passed to the transmitJSON api of the pshell, or other Popoto API (C++, Python, Matlab).

```
1 {
2   "ClassUserID": 0,
3   "ApplicationType": 2,
4   "Nationality": "ZX",
5   "O2": "17.000000",
6   "CO2": "5.000000",
7   "CO": "0.000000",
8   "H2": "5.000000",
9   "Pressure": "103.199997",
10  "Temperature": "50.000000",
11  "Survivors": "1",
12  "MobilityFlag": "1",
13  "ForwardingCapability": "1",
14  "TxRxFlag": "0",
15  "ScheduleFlag": "1",
16  "Schedule": "10",
17  "RepeatFlag": "1"
18 }
```

13.3.1.3 ApplicationType 3: Position + Status Message

Refer to the wiki page [Emergency Position and Status Message Format](#). The example below shows how to construct a representative ClassID 000 ApplicationType3 Position and Status Message for transmission by the Popoto Modem. This JSON structure would be encoded on a single line, and passed to the transmitJSON api of the pshell, or other Popoto API (C++, Python, Matlab).

```
1
2 {
3   "ClassUserID": 0,
4   "ApplicationType": 3,
5   "Nationality": "PT",
6   "Latitude": "38.386547",
7   "Longitude": "-9.055858",
8   "Depth": "16",
9   "Speed": "1.400000",
10  "Heading": "0.000000",
11  "O2": "17.799999",
12  "CO2": "5.000000",
13  "CO": "76.000000",
14  "H2": "3.500000",
15  "Pressure": "45.000000",
16  "Temperature": "21.000000",
17  "Survivors": "43",
18  "MobilityFlag": "1",
19  "ForwardingCapability": "1",
20  "TxRxFlag": "0",
21  "ScheduleFlag": "0"
22 }
```

13.3.2 Class User Id: 002 Underwater AIS

Refer to the wiki page [Underwater AIS Message Format](#) for details about the fields associated with the Underwater AIS message. The example below shows how to construct a representative ClassID 002 ApplicationType 8 underwater AIS message for transmission by the Popoto Modem. Note this AIS message has 2 contacts. This JSON structure would be encoded on a single line, and passed to the transmitJSON api of the pshell, or other Popoto API (C++, Python, Matlab).

```
1 {
2   "ClassUserID": 2,
3   "ApplicationType": 8,
4   "MobilityFlag": 1,
5   "Schedule": 0,
6   "TxRxFlag": 1,
7   "ForwardingCapability": 0,
8   "Contacts": [
9     {
10      "UserID": "0",
11      "Type": "15 =n.a.",
12      "Latitude": "-89.999001",
13      "Longitude": "0",
14      "Depth": "0.000000",
15      "Speed": "0.000000",
16      "Heading": "180",
17      "NavigationalStatus": "15 = Undefined/default"
18    },
19    {
20      "UserID": "1073741823",
21      "Type": "9 = Bottom node",
22      "Latitude": "-89.6485",
23      "Longitude": "-0.351552",
24      "Depth": "0.000000",
25      "Speed": "0.000000",
26      "Heading": "180",
27      "NavigationalStatus": "0 = Under way, using engine"
28    }
29  ]
30 }
```

13.3.3 Class User Id: 016 NATO JANUS reference

Refer to [NATO Janus Reference implementation](#) for description of the application types and the fields in the NATO Janus Reference Class ID. This Class is used for underwater chat applications, or for transmission of other short messages. Application Type 0 does not have a CRC to ensure delivery, while ApplicationType 001 validates the message with a CRC16.

Application Type 0

```
1 {
2   "ClassUserID": 16,
3   "ApplicationType": 0,
4   "StationID": "4",
5   "MobilityFlag": "1",
6   "ForwardingCapability": "1",
7   "DestinationID": "0",
8   "ParameterSetID": "0",
9   "Payload_Size": "16",
10  "Payload": {"Data": [49, 50, 51, 52, 53, 54, 55, 56, 57, 48, 49, 50, 51, 52,
11                    53, 54]}
```

Application Type 1

```
1 {
2   "ClassUserID": 16,
3   "ApplicationType": 1,
4   "StationID": "4",
5   "MobilityFlag": "1",
6   "ForwardingCapability": "1",
7   "TxRxFlag": "1",
8   "AckRequest": "1",
9   "DestinationID": "0",
10  "ParameterSetID": "0",
11  "Payload_Size": "16",
12  "Payload": {"Data": [ 49, 50, 51, 52, 53, 54, 55, 56, 57, 48, 49, 50, 51,
13                    52, 53, 54]}
```

14 Phorcys Operation

14.1 Phorcys System Configuration

Phorcys is a configurable underwater acoustic modem supporting flexible modulation, addressing, and security parameters. This chapter describes the `pshell` interface commands and the waveform reference tables required to configure and operate the modem.

All system variables described in Table 14.1.2 can be queried or set at run-time via the commands:

- `get <var>` — returns the current value
- `set <var>` — updates the value

14.1.1 Transmission Primitives

The modem can be instructed to transmit a Phorcys packet in two equivalent ways:

- **Alias command:** `phorcys <MS> <string>`
- **JSON command:**

```
transmitJSON {"PhorcysMS":<MS>, "Payload":{"Data":<string> or <data array>}}
```

14.1.2 System Variable Summary

Variable	Type	R/W	Min	Max	Description
PHOR_Mode	int	R/W	0	1	Selects modulation band: 0 (<i>Stheno</i> , 24 kHz) or 1 (<i>Euryale</i> , 10 kHz). Setting this variable automatically updates PHOR_Carrier.
PHOR_Carrier	int	R/W	5000	44000	Acoustic carrier frequency in Hz. Normally controlled by PHOR_Mode.
PHOR_PhyAddrTx	int	R/W	0	127	Transmit (remote) physical address.
PHOR_PhyAddrRx	int	R/W	0	127	Receive (local) physical address.

PHOR_SpCodeNo	int	R/W	0	7	Outer spreading-code index. Advanced users may replace the default code set with a custom OuterCodeFile.dat in /opt/popoto.
PHOR_Version	int	R/W	0	1	Polynomial version selector.
PHOR_PCIS	int	R/W	0	1	Encryption enable (1 = on).
PHOR_Seed	int	R/W	0	16777216	Seed value used when encryption is enabled.

14.1.3 Short-Payload (SP) Waveform Specification

Table 14.2: PHORCYSv0 – Short Payload (SP) waveform specification summary.

MS	MS bits				Payload (bits)	BT	Mod	Stheno (8kHz)			Euryale (4kHz)		
	b_3	b_2	b_1	b_0				Raw	Eff	Dur	Raw	Eff	Dur
0	0	0	0	0	8	16	16OS-QPSK	3000	200	40	1500	100	80
4	0	1	0	0	16	20	16OS-QPSK	3000	364	44	1500	182	88
8	1	0	0	0	24	24	16OS-QPSK	3000	500	48	1500	250	96
12	1	1	0	0	32	22	16OS-QPSK	3000	640	50	1500	320	100
1	0	0	0	1	8	16	16OS-QPSK	1500	167	48	750	83	96
5	0	1	0	1	16	20	16OS-QPSK	1500	286	56	750	143	112
9	1	0	0	1	24	24	16OS-QPSK	1500	375	64	750	188	128
13	1	1	0	1	32	22	16OS-QPSK	1500	471	68	750	235	136
2	0	0	1	0	8	16	64-OS	750	125	64	375	63	128
6	0	1	1	0	16	20	64-OS	750	200	80	375	100	160
10	1	0	1	0	24	24	64-OS	750	250	96	375	125	192
14	1	1	1	0	32	22	64-OS	750	308	104	375	154	208
3	0	0	1	1	8	16	64-OS	375	83	96	188	42	192
7	0	1	1	1	16	20	64-OS	375	125	128	188	63	256
11	1	0	1	1	24	24	64-OS	375	150	160	188	75	320
15	1	1	1	1	32	22	64-OS	375	182	176	188	91	352

14.1.4 Long-Payload (LP) Waveform Specification

Table 14.3: PHORCYSv0 – Long Payload (LP) waveform specification summary.

MS	MS bits				Payload (bytes)	BT	Mod	Stheno (8kHz)			Euryale (4kHz)		
	b_3	b_2	b_1	b_0				Raw	Eff	Dur	Raw	Eff	Dur
16	0	0	0	0	16	128	16OS-QPSK	3000	1103	116	1500	552	232

20	0	1	0	0	32	256	16OS-QPSK	3000	1620	158	1500	810	316
24	1	0	0	0	64	512	16OS-QPSK	3000	1803	284	1500	901	568
28	1	1	0	0	128	1024	16OS-QPSK	3000	1910	536	1500	955	1072
17	0	0	0	1	16	128	16OS-QPSK	1500	640	200	750	320	400
21	0	1	0	1	32	256	16OS-QPSK	1500	901	284	750	451	568
25	1	0	0	1	64	512	16OS-QPSK	1500	955	536	750	478	1072
29	1	1	0	1	128	1024	16OS-QPSK	1500	985	1040	750	492	2080
18	0	0	1	0	16	128	64-OS	750	348	368	375	174	736
22	0	1	1	0	32	256	64-OS	750	478	536	375	239	1072
26	1	0	1	0	64	512	64-OS	750	492	1040	375	246	2080
30	1	1	1	0	128	1024	64-OS	750	500	2048	375	250	4096
19	0	0	1	1	16	128	64-OS	375	182	704	188	91	1408
23	0	1	1	1	32	256	64-OS	375	246	1040	188	123	2080
27	1	0	1	1	64	512	64-OS	375	250	2048	188	125	4096
31	1	1	1	1	128	1024	64-OS	375	252	4064	188	126	8128

The SP formats prioritize brevity and high integrity for command-and-control traffic (1–4 bytes). LP formats sacrifice some compactness in exchange for Reed–Solomon forward-error correction and payloads up to 128 bytes, reaching effective throughputs within two-thirds of the raw rate.

14.1.5 Distinguishing Phorcys from Popoto Modulation on Reception

You can tell the difference between a received Popoto message and a Phorcys message based on the header. With a Phorcys message, the header is all zeroes except for the first digit, which will indicate the Phorcys modulation you are receiving. Standard Popoto messages will never have this pattern.

15 OEM Interface Description

15.1 Popoto Digital Interface

15.1.1 Overview

The Popoto Digital Interface (PDI) is a single connector which provides access to the most commonly used interfaces in the Popoto Modem system. These interfaces include RS-232, RS-422, 10/100 Ethernet, Board On/Off control, and PPS input signal.

15.1.2 PDI Hardware Components

PDI is connected to using a Molex Microfit connector (P/N 0430251400) or equivalent. This connector is sold as a shell plus discrete pins. While Molex produces many different pins for use with the MicroFit series, the best pins for use with Popoto Modems are Molex part number 0462355001. These pins are gold plated, rated for 250 mating cycles, and have a low insertion force. They are suitable for use with 20-24Ga wire. These pins can be crimped using one of Molex hand crimp tools such as the 0638190000. Alternately, if the expense of the crimp tool is cost-prohibitive for small prototype or limited production runs, pre-crimped wires are available from suppliers such as [Digikey](#).

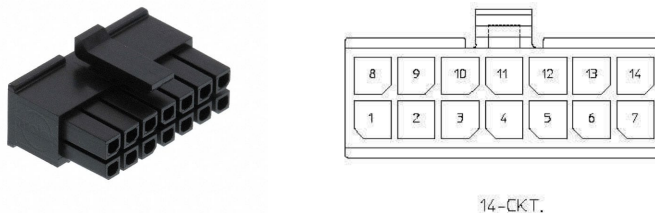


Figure 15.1: PDI User-Side Molex Connector. Interfacing to the PDI is accomplished with a Molex Microfit shell P/N 0430251400 and either Pre-pinned jumper wires, or Molex socket crimps.

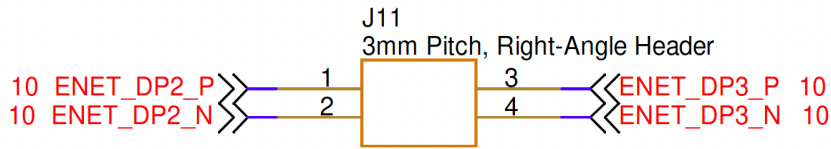


Figure 15.2: Additional PDI Connection to enable Gigabit Ethernet.

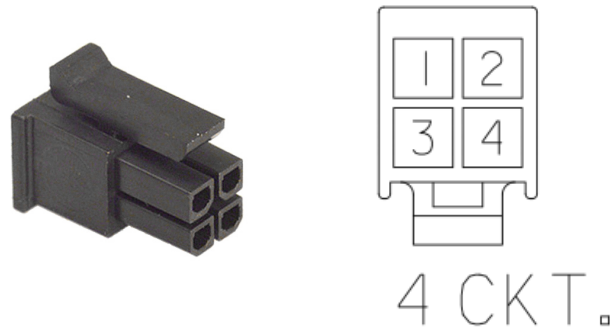


Figure 15.3: PDI Extended Gigabit User-Side Molex Connector. By Adding the 2 additional differential pairs, the Extended Gigabit connector enables gigabit ethernet to the PMM6081.

15.1.2.1 PDI Expanded Gigabit Ethernet

In addition to the standard Ethernet capabilities, the PMM6081 boasts an advanced feature for enhanced network performance. A separate 4-pin port (see Figure ?? and Figure ??) is included on the device, specifically designed to enable Gigabit Ethernet. This port, when used in conjunction with the 14-pin connector's four Ethernet pins, unlocks the full potential of Gigabit Ethernet speeds. This feature is particularly beneficial for applications that require high-speed data transfer, such as large file transfers, multichannel audio streaming, or rapid upgrades. .

15.1.3 Electrical Connections

Figure 15.4 shows the electrical connections for the the PDI.

- Pins labeled RS-422 are UART signals that comply with EIA-RS-422 interface standards.
- Default UART signaling parameters are 115200N81.
- Pins labeled with RS-232 are UART signals that comply with EIA-RS-232 electrical interface standards.
- UART signaling parameters for the RS-232 port default to 115200N81.
- Power_OFF_N allows the unit to be powered off by connecting this signal to ground.
- ENET Signals are 10 100 Ethernet signals. As the Popoto board has on-board magnetics, these signals are standard 10 100 BaseT Ethernet signals.
- PpsInput is a 3.3V logic level input signal that is used for PPS input for clock discipline.

Table 15.1: PDI Components and Part Numbers

Part Number	Manufacturer	Description
0430251400	Molex	Microfit 14 position connector Receptacle 3.0MM
0430250400	Molex	Microfit 4 position connector Receptacle 3.0MM for Gigabit Extended PDI
0462355001	Molex	Microfit 20-24Ga gold plated, lubricated sockets
0638190000	Molex	Microfit Hand Crimp tool
0797580010	Molex/Digikey	Precrimped Microfit leads

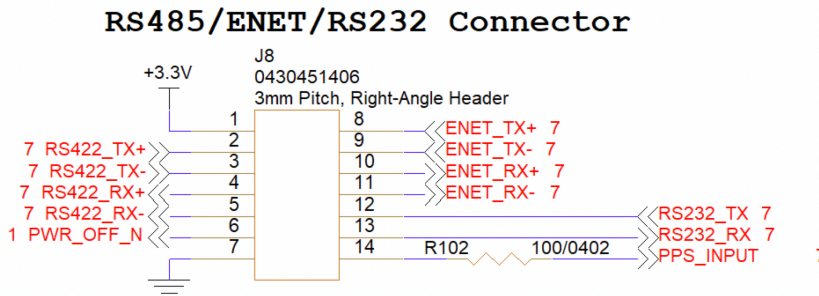


Figure 15.4: PDI Schematic connections.

Table 15.2: PDI Electrical Pinout

Pin Number	I/O	Pin Name	Notes
1	O	3.3V	3.3V out when unit is powered up
2	O	RS 422 Tx +	Connect to Rx+ on Host
3	O	RS 422 Tx -	Connect to Rx- On Host
4	I	RS 422 Rx+	Connect Tx+ on Host
5	I	RS 422 Rx-	Connect to Tx- on Host
6	I	PowerSwitch	Short to ground to power down unit
7	-	Gnd	Digital Ground
8	O	Ethernet Tx+	T568A Green White T568B Orange White
9	O	Ethernet Tx-	T568A Green T568B Orange
10	I	Ethernet Rx+	T568A Orange & White T568B Green & White
11	I	Ethernet Rx-	T568A Orange T568B Green
12	O	RS-232 TX	Connect RX on Host
13	I	RS-232 RX	Connect to Tx On Host
14	I	PPS Interrupt	PPS interrupt for optional time Sync Max Voltage 3.3V for PMM3511 5V for PMM5021

The PMM6081 provides an additional Connector (J11) to enable gigabit ethernet. To use the gigabit ethernet, wire the PDI port per table 15.2 and add the additional 2 differential pairs for the ethernet as shown below.

Table 15.3: Extended Gigabit PDI Electrical Pinout

Pin Number	I/O	Pin Name	Notes
1	O	Ethernet Diff Pair 2P	
2	O	Ethernet Diff Pair 2N	
3	I	Ethernet Diff Pair 3P	
4	I	Ethernet Diff Pair 3N	

15.1.4 Digital Interfaces

Popoto Modems have 3 additional digital interfaces beyond the PDI port. These interfaces are used to connect to external devices, or to provide alternate digital connection schemes for a host controller.

UART (3.3V) Port Connector

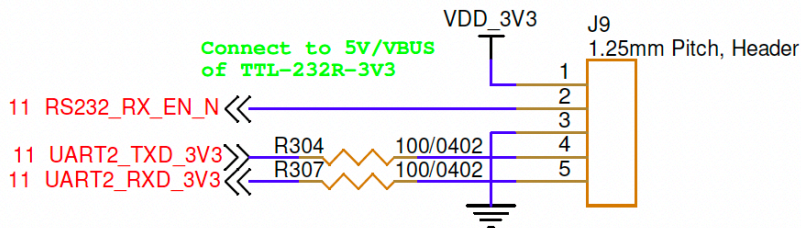


Figure 15.5: Popoto TTL UART Plug. This port allows 3.3V logic-level UART connections

15.1.4.1 TTL Uart

The TTL UART port is used for connecting Popoto to a local controller over a short distance. The TTL UART port is a 5 pin Molex picoblade connector. Figure 15.5 shows the schematic connections on the TTL-UART port. In order to enable the 3.3V uart port, pins one and 2 of J6 must be shorted together. Doing this disables the RS-232 level translator, and thereby disables the RS232 port on the PDI connector.

Table 15.4: Popoto TTL UART Parts

Part Number	Manufacturer	Description
0510210500	Molex	Picoblade 5 position connector Receptacle
0500798000	Molex	Picoblade 26-28Ga sockets
2002181900	Molex	HAND TOOL FOR PICO-BLADE 26-32AW
2149202214	Molex	Precrimped Picoblade 150mm 26Ga

Table 15.5: Popoto 3.3V Uart Port

Pin Number	I/O	Pin Name	Notes
1	P	V+	+3.3V
2	I	V+	RS232_EN_N Tie this pin high (short to pin 1) to enable the 3.3V UART port
3	G	GND	Ground
4	O	UART0_TXD	Popoto UART Output
5	I	UART0_RXD	Popoto UART Input

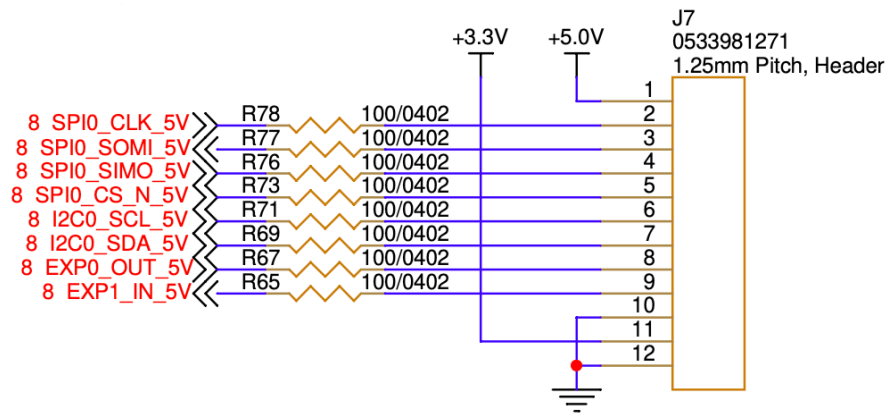


Figure 15.6: Popoto Expansion Header. This connector allows access to I2C, SPI and General purpose I/O from the Popoto Modem.

15.1.4.2 Digital Expansion Header

Figure 15.6 shows the schematic for the Digital Expansion Header. This header enables peripheral access when applications execute directly on the Popoto Modem's System on a Chip (SOC). It features general-purpose input/output (GPIO) pins, along with SPI and I2C interfaces. Furthermore, signals from this connector are utilized for Push-to-Talk (PTT) and volume control within SSB mode. The connector is a 12-pin Picoblade type, and the necessary components are listed in Table 15.6

Table 15.6: Popoto Expansion Header Parts

Part Number	Manufacturer	Description
0510211200	Molex	Picoblade 12 position connector Receptacle
0500798000	Molex	Picoblade 26-28Ga sockets
2002181900	Molex	HAND TOOL FOR PICO-BLADE 26-32AW
2149202214	Molex	Precrimped Picoblade 150mm 26Ga

15.1.4.3 0056 Analog Board GPIO Expansion Header

On the PMM6081 and PMM5544 boards shipped with the 068-0056-xx version analog boards, a GPIO header is provided for additional interface possibilities. The connector is found at J5 along the edge of the board and is a Molex 12 pin Picoblade connector. Refer to 15.7 for part numbers for this part, and refer to 15.7 for the schematic diagram. This pins are accessible from the Linux GPIO subsystem.

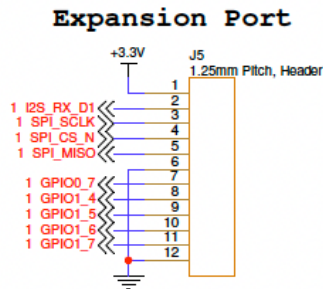


Figure 15.7: Popoto 0056 Board Expansion Header. This connector allows access to I2C, and General Purpose I/O (GPIO) from the Popoto Modem.

Table 15.7: 0056 Analog Board GPIO Expansion Header

Part Number	Manufacturer	Description
0510211200	Molex	Picoblade 12 position connector Receptacle
0500798000	Molex	Picoblade 26-28Ga sockets
2002181900	Molex	HAND TOOL FOR PICO-BLADE 26-32AW
2149202214	Molex	Precrimped Picoblade 150mm 26Ga

15.1.4.4 MCU Expansion Header

The MCU Expansion header allows interface to the Popoto wake up processor. The Popoto wakeup processor is a mixed signal device. This device has Analog inputs, as well as digital I/O at 1.8V. This port is especially useful for monitoring signals while the main processor is in Deep sleep mode. Use of this port requires special firmware support from Popoto Modem. If you require access to these signals for your application, please reach out to info@popotomodem.com.

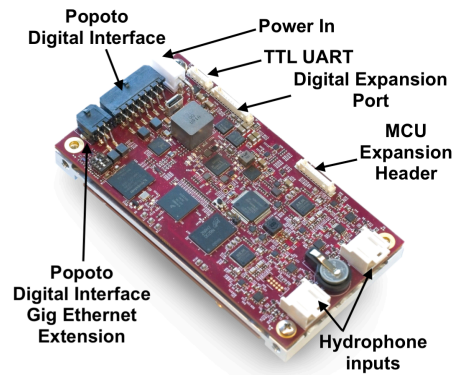


Figure 15.8: PMM6081 Digital Board Connector Locations

15.1.4.5 Micro USB Port

The Micro USB port is a standard USB OTG port as configured by the Popoto Modem Linux Operating system. This port is extremely flexible, allowing both host and peripheral connections. If you have need for the Micro USB port, please contact Popoto Modem at info@popotomodem.com.

15.2 PMM6081 Specific Interfaces

15.3 PMM6081 Specific Interfaces

15.3.1 Power

Power is provided to the PMM6081 OEM Boardset via connector J1 on the Digital Board. This connector is a 2 pin Molex MiniFit Jr connector, and has provisions for V+ pin and Ground pin. Acceptable input voltages are between 8.5 and 36 Volts. Table 15.9 and Figure 15.9 show the connections required for powering the PMM6081. Table 15.8 shows the parts required for attaching to the power connector on the PMM6081. Two option are given: Using sockets and a crimp tool for larger production runs, or ordering precrimped wires from Digikey for smaller prototype/production runs.

Table 15.8: PMM6081 Power Plug Components

Part Number	Manufacturer	Description
0039013022	Molex	MiniFit Jr 2 position connector Receptacle
0039000182	Molex	MiniFit Jr 18-24Ga gold plated, sockets
0638190901	Molex	Minifit Hand Crimp tool
0039000038-12-R9	Molex/Digikey	Precrimped MiniFit 12in 18Ga Red
0039000038-12-K9	Molex/Digikey	Precrimped MiniFit 12in 18Ga Black

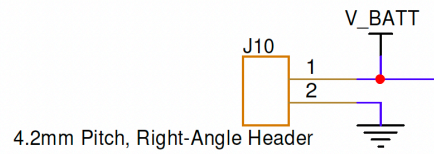


Figure 15.9: PMM6081 Power Schematic.

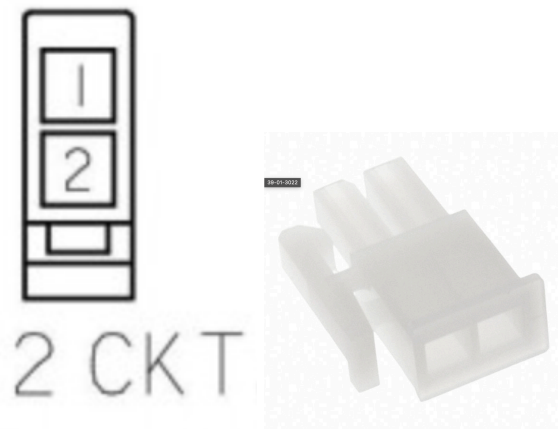


Figure 15.10: PMM6081 Power Connectors and pinout.

Table 15.9: PMM6081 Power Connector Pinout

Pin Number	I/O	Pin Name	Notes
1	P	V+	8.5-36 Volts 150 Watts
2	G	GND	Ground

15.3.2 Analog Interfaces

15.3.2.1 Hydrophone Interfaces for PMM6081 Digital Board

The PMM6081 Digital Board features two 10 Pin Molex iGrid connectors for hydrophone interfaces. These connectors are designated as J1 and J2, each providing different levels of gain for hydrophone channels.

15.3.2.2 J1: High Gain Hydrophone Connector

The J1 connector provides a High Gain 43dB amplification suitable for hydrophone channels. The pinout configuration for the J1 connector is as follows:

Pin	Function	Description
1	AIN4P	Analog input 4 Positive
2	AIN4N	Analog input 4 Negative
3	AIN3P	Analog input 3 Positive
4	AIN3N	Analog input 3 Negative
5	AIN2P	Analog input 2 Positive
6	AIN2N	Analog input 2 Negative
7	AIN1P	Analog input 1 Positive
8	AIN1N	Analog input 1 Negative
9	VSYS5V	5V output
10	GND	Ground

Table 15.10: J1 High Gain Hydrophone Connector Pinout, AINxP and AINxN are the positive and negative inputs for a given hydrophone channel.

15.3.2.3 J2: Low Gain Hydrophone Connector

The J2 connector provides a Low Gain (12dB) for hydrophone connections. The pinout configuration for the J2 connector is as follows:

Pin	Function	Description
1	AIN8P	Analog input 8 Positive
2	AIN8N	Analog input 8 Negative
3	AIN7P	Analog input 7 Positive
4	AIN7N	Analog input 7 Negative
5	AIN6P	Analog input 6 Positive
6	AIN6N	Analog input 6 Negative
7	AIN5P	Analog input 5 Positive
8	AIN5N	Analog input 5 Negative
9	VSYS5V	5V output
10	GND	Ground

Table 15.11: J2 Low Gain Hydrophone Connector Pinout, AINxP and AINxN are the positive and negative inputs for a given hydrophone channel.

15.3.2.4 Hydrophone Connectors Part Number Table

Component	Manufacturer Part Number
Crimps	Molex 5016471000
Housing	Molex 5016461000

Table 15.12: Manufacturer Part Number for iGrid Pins and Shells

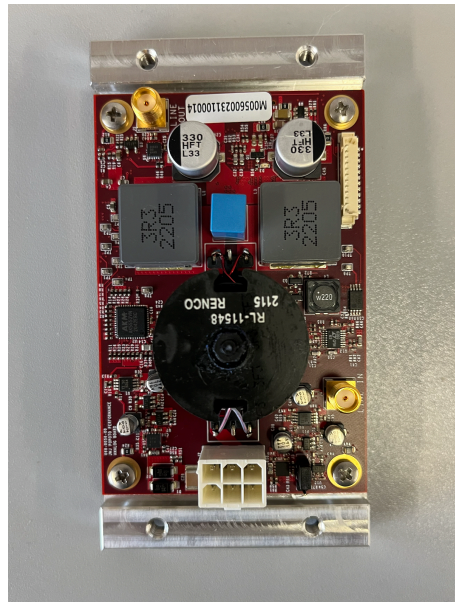


Figure 15.11: The PMM6081 Analog board

15.3.3 Analog Board

The remaining analog interfaces to the PMM6081 can be found on the analog board. This board has the large round pot-core inductor, and can be seen in [Figure 15.11](#)

15.3.3.1 Transducer

The Transducer is connected to the Popoto Modem by a 6 pin Molex MiniFit Jr connection, labelled J9. This connector provides access to the TPA output and provides positions for series and parallel matching networks. In its default configuration with the Popoto 25-30Khz transducer, no additional matching networks are required. See [Figures 15.12](#) and [15.13](#) for the pinout for this connector.

**Place Series Matching network from Pin 1 to 2.
 Short with a shorting loop if no match needed
 Place parallel Matching network from 6 to 5.
 Connect transducer to pins 6 (inner ring) and 4 (Outer Ring)**

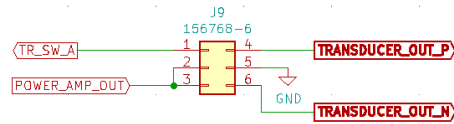


Figure 15.12: The PMM6081 Transducer connector schematic.

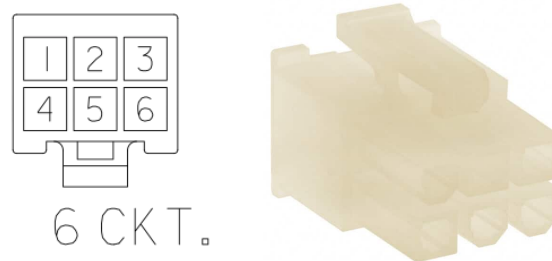


Figure 15.13: PMM6081 Transducer connector and pinout.

Table 15.13: PMM6081 Transducer Connector Pinout

Pin Number	I/O	Pin Name	Notes
1	I	TR_SW_A	Input to the TR Switch. Connect to Pin 2 with series matching network
2	O	POWER_AMP_OUT	Connect to Pin 1 with series Matching network
3	O	POWER_AMP_OUT	Same signal as Pin 2
4	O	TRANSDUCER_OUT_P	Positive transducer connection. Connect to Pin 5 with parallel matching network if needed
5	G	GND	Ground
6	O	TRANSDUCER_OUT_N	Negative transducer connection.

Table 15.14: PMM6081 Transducer Plug Parts

Part Number	Manufacturer	Description
0039012060	Molex	MiniFit Jr 6 position connector Receptacle
0039000182	Molex	MiniFit Jr 18-24Ga gold plated, sockets
0638190901	Molex	Minifit Hand Crimp tool
0039000038-12-R9	Molex/Digikey	Precrimped MiniFit 12in 18Ga Red
0039000038-12-K9	Molex/Digikey	Precrimped MiniFit 12in 18Ga Black

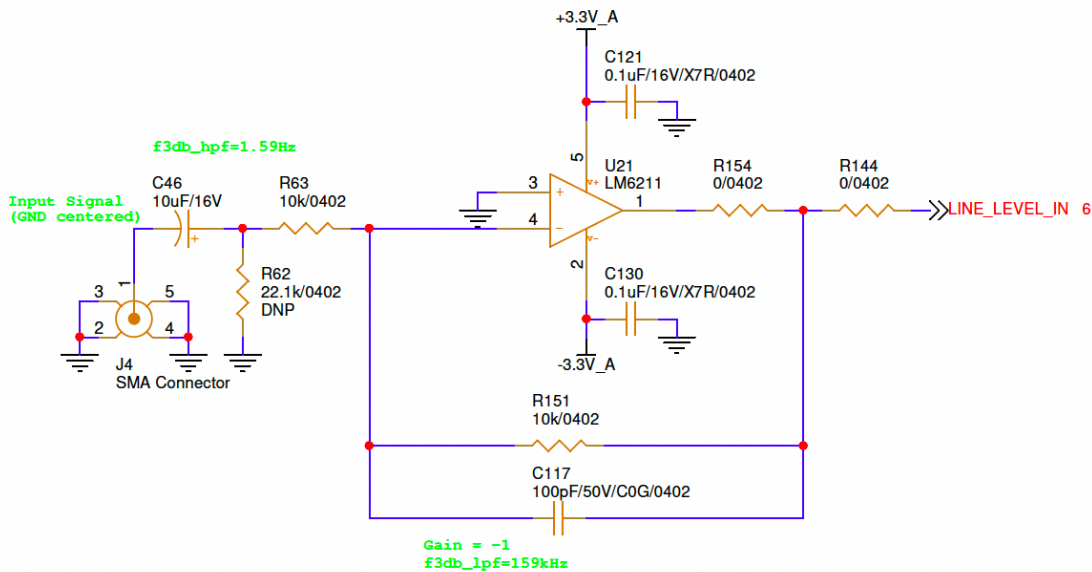


Figure 15.14: PMM6081 Analog input schematic excerpt. This circuit conditions the input signal and is used for SSB voice input or for applications providing line level analog input.

15.3.3.2 Analog In

The PMM6081 Analog board has provisions for analog input via an SMA connector mounted on the analog board. This connector is used for SSB voice input, as well as for applications that have line level outputs of transducer signals. The Analog input port drives an adjustable gain amplifier to allow for level matching between different equipment. An excerpt of the schematic, showing the input amplifier topology is shown in Figure 15.14. Note that for the analog input to operate, the J5 and J1 jumpers must be installed and J2 should be installed in the 2-3 position to connect the input to SMA. The input impedance of the SMA connection is 22.1 K. The input gain is adjustable by R13 yield a gain spanning from 1/2 to 25. The A/D input spans +/- 2.5 volts.

15.3.3.3 Analog Out

The PMM6081 Analog board has provisions for analog output via an SMA connector mounted on the analog board. This connector is used for SSB voice output, as well as for applications that utilize offboard power amplifiers. The Analog output port drives a fixed gain amplifier to provide buffering and level setting of the output. An excerpt of the schematic, showing the input amplifier topology is shown in Figure 15.15. The full scale output voltage on the SMA

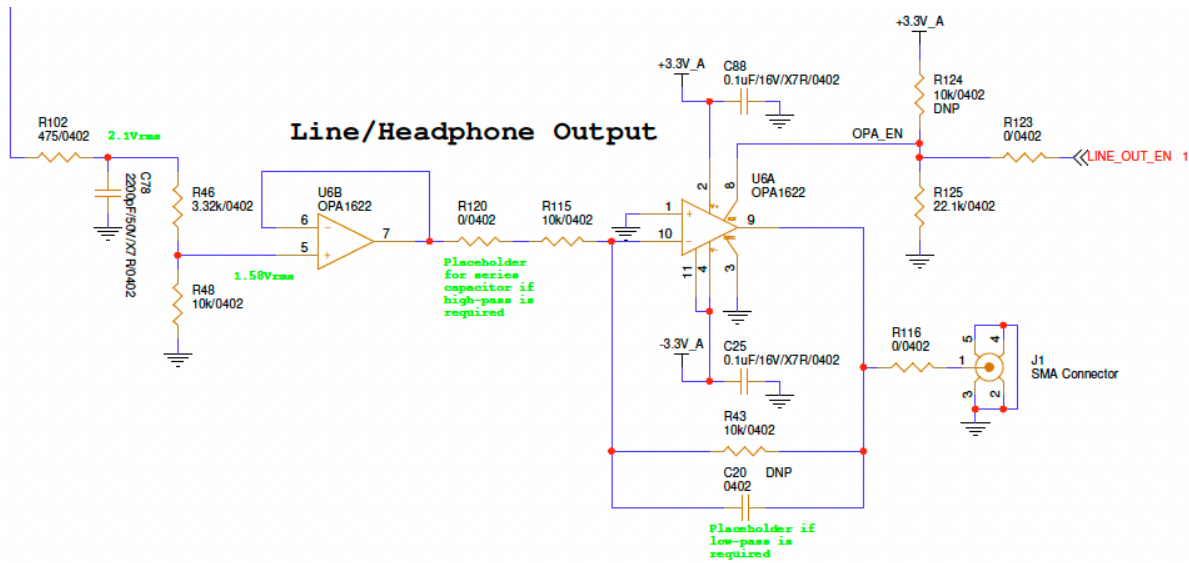


Figure 15.15: PMM6081 Analog output schematic excerpt. This circuit provides a +/- 3.3V signal to the SMA output port. This signal is used for the headphones output during SMA voice mode, or for a diagnostic port or to drive an external power amplifier if needed.

is +/- 3.3 Vpp. The maximum output current is 145mA and is ground centered. The 3dB cut off point of the output low-pass filter is 152KHz.

16 Battery Multiplexer

16.1 Battery Multiplexer

16.1.1 Overview

The Popoto Battery Multiplexer is a hardware component that allows use of multiple Lithium Ion battery packs in parallel. The Battery Multiplexer draws power from the most fully charged battery, and will draw that battery down until it is below the charge level of another battery, at which point it will seamlessly switch over to that one. Similarly, it allows bank charging of the batteries, charging the most drained battery first, and then switching to the others as they charge. Charging is accomplished with the standard Popoto Lithium Ion Charger. For wired installations, this will require an adapter cable from

16.1.2 Power Connectors

Table 16.1: Battery Multiplexer Power In and Out Plug Components

Part Number	Manufacturer	Description
0039013042	Molex	MiniFit Jr 4 position connector Receptacle
0039000182	Molex	MiniFit Jr 18-24Ga gold plated, sockets
0638190901	Molex	Minifit Hand Crimp tool
0039000038-12-R9	Molex/Digikey	Precrimped MiniFit 12in 18Ga Red
0039000038-12-K9	Molex/Digikey	Precrimped MiniFit 12in 18Ga Black

Table 16.2: Battery Mux Power Connector Pinout

Pin Number	I/O	Pin Name	Notes
1	P	V+	12-29 Volts 150 Watts
2	P	V+	12-29 Volts 150 Watts
3	G	GND	Ground
4	G	GND	Ground

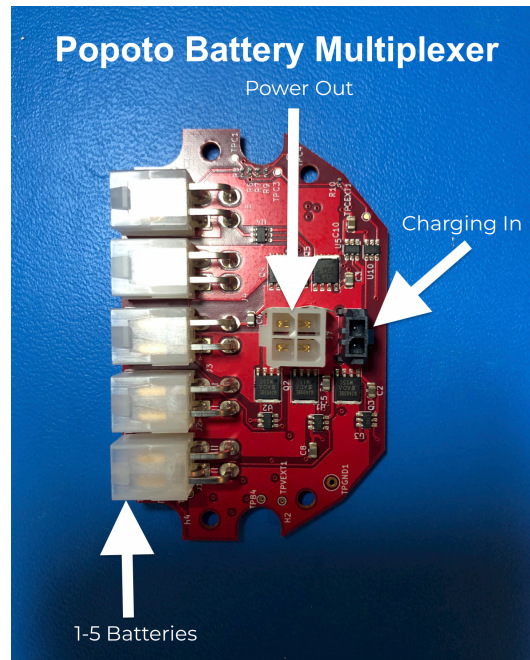


Figure 16.1: Popoto Battery Mux Board. Power is provided by 1-5 6S Lithium Ion Batteries provided on the Molex MiniFit Jr 4 pin connectors on the left. Power is drawn from the Minifit Jr 4 port connector on the top, and charging is accomplished by connecting the charger to the 2 port microfit connector on the right side of the diagram.

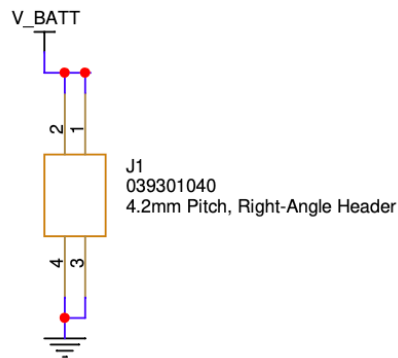


Figure 16.2: Battery Mux: Battery in Power out Schematic.

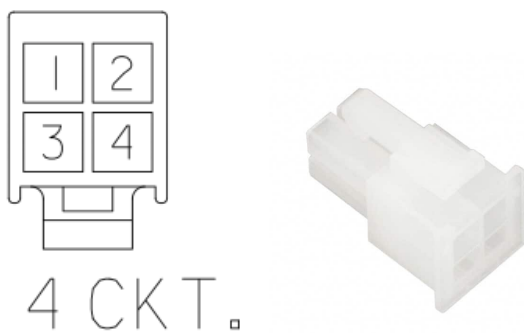


Figure 16.3: PMM5021 Power Connectors and pinout.

CHARGING PORT CC/CV 25.2 V

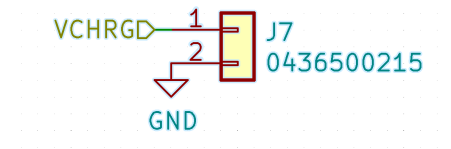


Figure 16.4: Battery Mux: Charging Port Schematic. Use a constant current/constant voltage lithium ion charger for 6S Battery Packs. (1.875A)

16.1.3 Charging Connectors

The charging connector, J7, on the Battery Multiplexer (see Figure 16.4) will charge all battery packs in parallel using the standard Popoto CC/CV charger.

Table 16.3: Battery Mux Charging Port Part Numbers

Part Number	Manufacturer	Description
0436500214	Molex	Microfit 2 position connector Receptacle 3.0MM
0462355001	Molex	Microfit 20-24Ga gold plated, lubricated sockets
0638190000	Molex	Microfit Hand Crimp tool
0797580010	Molex/Digikey	Precrimped Microfit leads

17 Popoto Interface Board

17.1 Popoto Interface Board

17.1.1 Overview

The Popoto interface board provides a simple way to connect a host computer to the Popoto Modems. The Popoto Interface board connects to the modem via the PDI, and the PC connects to the Popoto Interface Board via USB and ethernet.

17.1.2 PDI Connector J10

The PDI connector connects pin for pin to the PDI connector on the Popoto Modem. Pin 1 of J10 connects to pin 1 of the PDI, pin (J10) 2 to PDI pin 2, ...

17.1.3 USB Port (J2)

The Popoto Interface board connects via USB to the host computer. The USB Port enumerates as 2 serial ports. These ports will typically show up in Windows as COMn and COMn+1, and in Linux as TTYUSBn and TTYUSBn+1. Both serial ports are enabled on all M2000/M6000 devices. S1000 devices are ordered with either ethernet, RS-232 or RS-422. On these devices, only the enabled interface will be available.

Table 17.1: USB Ports

Port Number	Serial Protocol	Default terminal
PORT n+0	RS-422	Pshell
PORT n+1	RS-232	Linux Terminal

17.1.4 Ethernet Port (J1)

The Ethernet port (J1) provides a standard RJ-45 ethernet connection to the Popoto Modem. This port is active on all M2000 devices, and on S1000 devices that are configured for ethernet.

17.1.5 Switch SW1 and Jumper J9

SW1 provides an illuminated power on/off switch. To enable this switch jumper J9 must be installed. Note that the illumination of the switch will be delayed by 3-5 seconds after turn-on of the Popoto board.

17.1.6 PWRDN LED (J8)

The PWRDN LED connector is a 4 pin Molex microfit connector that allows the user to supply a remote illuminated switch. In order to use the remote switch, Jumper J9 must be removed.

Table 17.2: PWRDN LED Connector Pinout

Pin	I/O	Signal
1	P	MODEM 3.3V output voltage
2	I	PWDN N Signal: Connect to GND to power down
3	O	LED voltage.
4	P	GND

17.1.7 SSB Connections

The following connectors are used for Single Side Band Voice with the Popoto PMM5021 based devices. In order to use SSB voice, Connections J3, J4, J5, J6, and J7 are needed for SSB voice input and control.

Table 17.3: SMA Ports J6 and J7

Port Number	I/O	Signal	PMM5021 connection
J6	I	SSB Headphone	PMM5021 Analog Port J4
J7	O	SSB Microphone	PMM5021 Analog Port J3

Table 17.4: SSB Control Port J3

Pin	I/O	Signal	Operation
1	I	VOL+	Momentarily Ground to increment Volume
2	-	GND	
3	I	VOL-	Momentarily Ground to decrement Volume
4	I	PTT0	Momentarily Ground at the same time as PTT1 for Push to talk
5	-	GND	
6	I	PTT1	Momentarily Ground at the same time as PTT0 for Push to talk

Table 17.5: Headphone/Microphone Connector J5

Pin	I/O	Signal
1	O	Left Headphones
2	O	Right Headphone
3	I	Microphone
4	P	GND

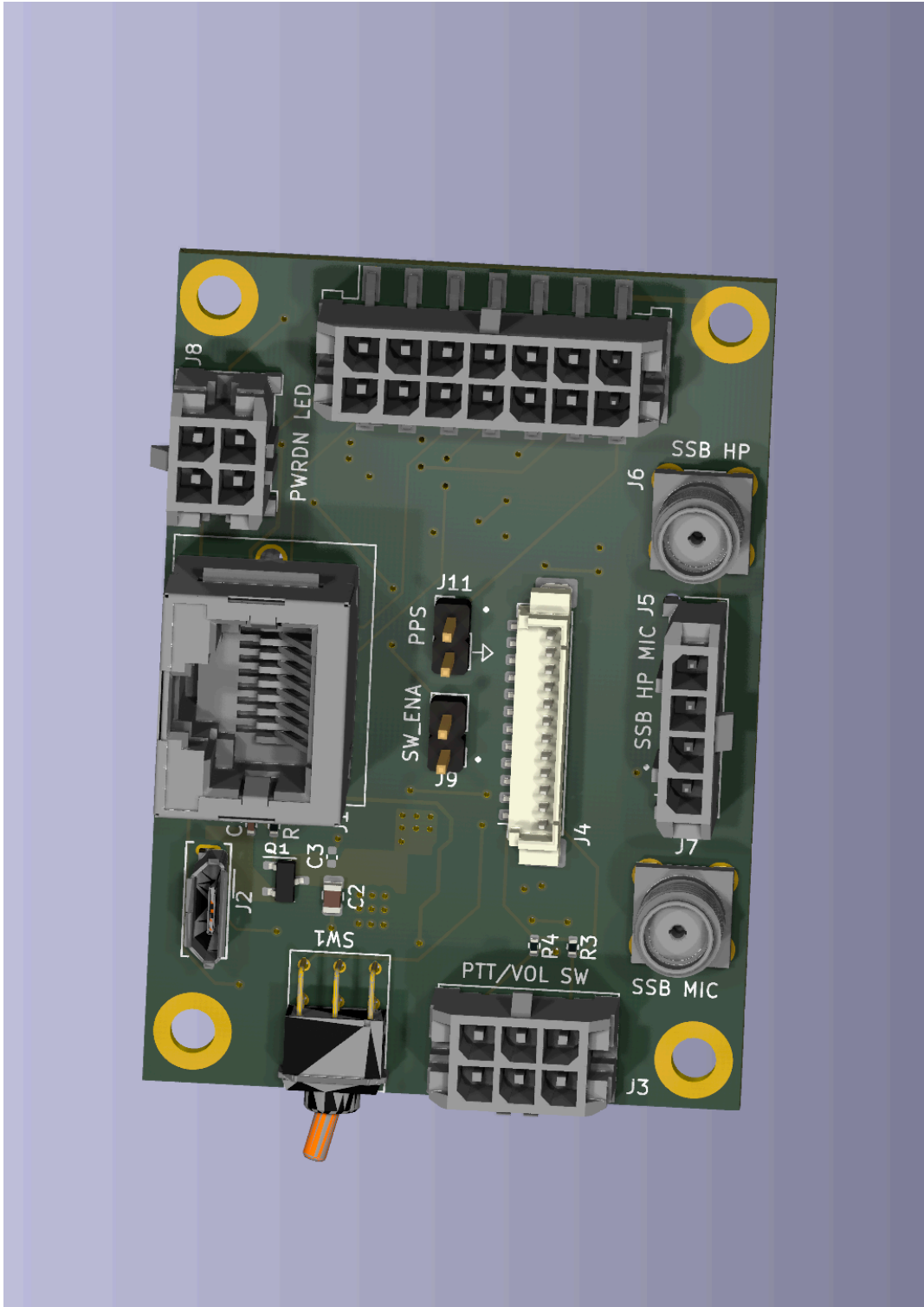


Figure 17.1: Popoto Interface Board

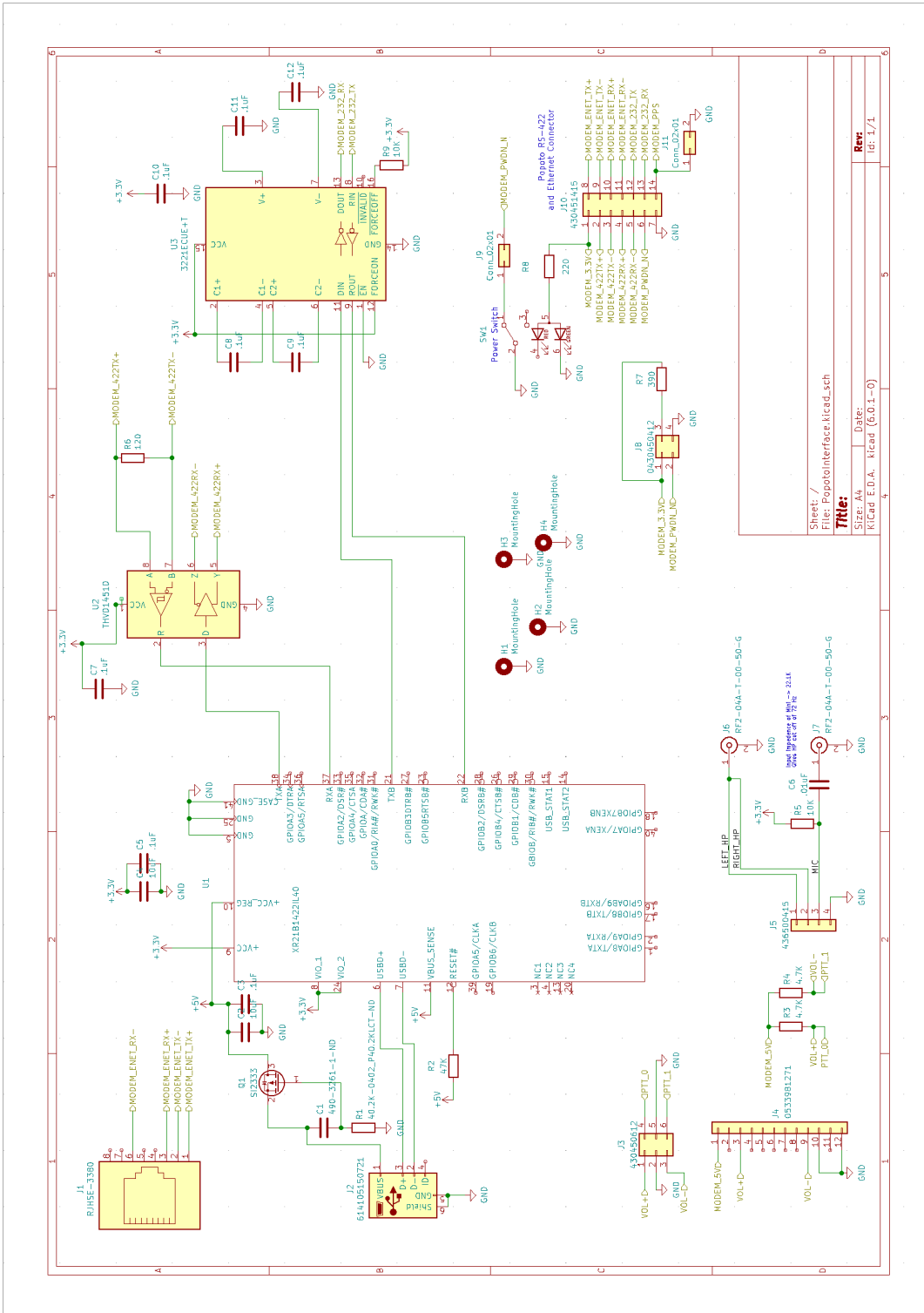


Figure 17.2: Popoto Interface Board Schematic

18 Upgrading the Firmware

18.1 Introduction

Firmware updates are accomplished through the ethernet port. As an overview the process involves three steps:

1. SCP an update tar file
2. Extract the tar file
3. Run the update shell script

18.1.1 Details on how to update the firmware

Requirements:

- Laptop or Desktop Computer
- Network Connection
- Pshell connection (Rs 422)
- Secure Shell/Copy utility
- Update_<version_num>.tar

18.2 Upload Procedure

Note: By default the Popoto boardset is shipped without a root password. the examples below all remote operations happen without password entry. If you have added a root password to your system, please enter the password when prompted by the SSH and SCP utilities in the steps below.

Step 1

Connect Popoto to ethernet connection.

Step 2

Determine or Set the Popoto IP Address From the pshell issue getIP

```
Popoto-> getIP
IPv4 Address: eth0 Link encap:Ethernet HWaddr 2E:A4:4D:D2:40:82
inet addr:10.0.0.232 Bcast:10.255.255.255 Mask:255.0.0.0
inet6 addr: 2603:3005:82a:8000:2ca4:4dff:fed2:4082%71/64 Scope:Global
inet6 addr: fe80::2ca4:4dff:fed2:4082%71/64 Scope:Link UP ...
RX packets:639 errors:0 dropped:0 overruns:0 frame:0 TX packets:15...
RX bytes:57888 (56.5 KiB) TX bytes:26702 (26.0 KiB) Interrupt:33
```

In this example the IP address is 10.0.0.232.

To change the IP address of the Popoto, issue the setIP command from pshell.

Step 3

Confirm connection to the Popoto's network connection using the ping command from your local computer's command window.

```
ping 10.0.0.232
PING 10.0.0.232 (10.0.0.232): 56 data bytes
64 bytes from 10.0.0.232: icmp\seq=0 ttl=64 time=0.853 ms 64 bytes from...
```

Step 4

Using a secure copy utility, such as OpenSSH's scp, located on your local computer, copy the update file to the Popoto's root directory

```
scp Update_2.7.0.tar root@10.0.0.232:/
```

Step 5

Shell into the Popoto, using an ssh utility

```
ssh root@10.0.0.232
```

You should receive a prompt like:

```
root@popoto:~#
```

Step 6

From the root@popoto: prompt, change directories to the root directory, and untar the update file previously uploaded.

```
root@popoto:~# cd /
root@popoto:/# tar xvf Update\_2.6.0.tar
```

This will create (or overwrite) the following 2 files

- Update.sh
- Update.tgz

Step 7

Execute the Update shell command to install the newest version.

```
root@popoto:/# ./Update.sh
```

This will generate the output similar to.

```
Update.sh
Version.txt
boot/
boot/dolphin.dtb
home/
home/root/
home/root/popoto.py
home/root/pshell
home/root/popoto\_app
lib/
lib/firmware/
lib/firmware/platform.out
pshell.init
version.txt
Connection to 10.0.0.232 closed by remote host. Connection to 10.0.0.232 closed.
```

At this point the Popoto unit will reboot, and come up with the new firmware version installed.

In the pshell window (RS-422) you should end up at a prompt that says

```
        Welcome to the Popoto modem Shell! Communicating Naturally
Popoto-> {"Info ":"Popoto Modem Version <New Version Number and informational tag> "}
```

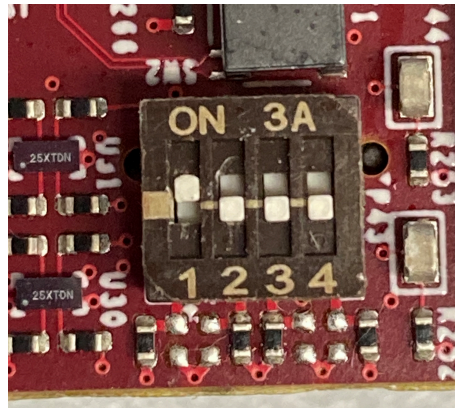


Figure 18.1: PMM6081 Switch Position used for running program command

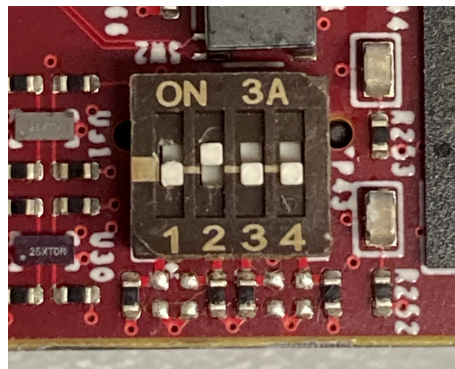


Figure 18.2: PMM6081 Switch Position used for running program command

18.3 Updating Image using UUU

18.3.0.1 Instructions for for Ubuntu 22.04

To reprogram the EMMC flash on a PMM6081 board follow the procedure below:

First set the dip switches as shown in [18.2](#) and run the following command:

```
sudo uuu -v -b emmc_all ./imx-boot ./pmm6081-image-pmm6081.wic
```

To program a board on a specific USB port, use the -m argument, followed by the USB port:

```
sudo uuu -m 1:1 -v -b emmc_all ./imx-boot ./pmm6081-image-pmm6081.wic
```

When done, reset the dip switches to the :

NOTE: This will be slow! 30 min, follow instructions below to get a faster update utility

18.3.0.2 Fast UUU Upload

This section covers the procedure and tools required to program a compressed archive. These archives are distributed as a wic.bz2 file along with a bmap. Many distributions do not include a pre-built uuu, part of the mfgtools package, capable of flashing a bz2 file. So we will need to build one from source.

18.3.0.3 Setup

We will obtain the proper version mfgtools source from the following website and commit hash

```
https://github.com/nxp-imx/mfgtools commit hash: `dc3ca54`
```

Using the following procedure obtain the source code from the repository.

```
mkdir ~/fast_uuu
cd ~/fast_uuu
git clone https://github.com/nxp-imx/mfgtools.git
cd mfgtools
```

Install all required dependencies for the mfgtools build.

```
sudo apt-get install libusb-1.0-0-dev libbz2-dev libzstd-dev pkg-config cmake libssl-dev g++ zli
```

Once this completes, we will execute the build commands

```
cmake . && make
```

After the build is complete, copy the executable from the uuu subdirectory to your user software folders:

```
cd ~/fast_uuu/mfgtools/uuu/
sudo cp uuu /usr/bin/uuu
sudo cp uuu /usr/local/bin/uuu
```

This enables using the uuu command from any directory.

18.3.1 Updating with fast uuu

To update using the new fast uuu, use the following command:

```
sudo uuu -bmap -v -b emmc_all /path/to/imx-boot path/to/pmm6081-image-pmm6081.wic.bz2
```

NOTE: In order for fast flashing to work, you must use the .wic.bz2 file.

19 Network Settings

19.1 Adjusting Network Settings on Popoto Debian

The network on the Popoto Modem operating system is configured to use a static IP address and a MAC address that is randomly generated when the system boots. To change the IP address, you can use the pshell's setIP command (see in Pshell Command Reference below).

In the Debian OS, the file `/etc/init.d/staticip` is used to set the static IP address of the modem. The file should look like this on the inside:

```
#!/bin/sh
NAME=staticip
DESC="Configure Eth0 Static"

case "$1" in
  start)
    echo -n "$DESC: "
    /sbin/sysctl -w net.ipv4.igmp_max_memberships=10
    /sbin/ifconfig eth0 10.0.0.237
    /sbin/ifconfig eth0 up &
    /bin/mount -a
    ;;
  stop)
    echo -n "Stopping $DESC: "
    /sbin/ifdown eth0
    ;;
  restart|force-reload)
    ;;
esac

exit 0
```

To set a static MAC address, you should modify this file such that the MAC address is specified immediately after the IP address:

```
#!/bin/sh
NAME=staticip
```

```
DESC="Configure Eth0 Static"

case "$1" in
  start)
    echo -n "$DESC: "
    /sbin/sysctl -w net.ipv4.igmp_max_memberships=10
    /sbin/ifconfig eth0 10.0.0.237
    /sbin/ifconfig eth0 hw ether 02:XX:XX:XX:XX:XX
    /sbin/ifconfig eth0 up &
    /bin/mount -a
    ;;
  stop)
    echo -n "Stopping $DESC: "
    /sbin/ifdown eth0
    ;;
  restart|force-reload)
    ;;
esac

exit 0
```

The MAC address should begin with 02: or 06: and contain randomly-generated bytes (in hexadecimal format, from 00 to FF) in place of the XX's above.

20 Diagnostics

The FOAM architecture has built in logging support to enable diagnostics and debug of any in field problems. The logging consists of a rolling file based log file, along with options for saving the passband PCM data. The log file is useful for determining message flow and state transitions, and the PCM passband logging is useful for diagnosing signal processing and signal quality issues.

20.1 Popoto log

20.1.1 Introduction

The Popoto.log is a diagnostic logfile which is updated as the Popoto_app runs, keeping track of message and logic flows within the system. This logfile has the following properties.

- The Log file is Leveled: All logs are assigned a severity level in the code, and by changing the output filter, only logs greater than a set severity level are displayed.
- The log file is Timestamped: Each log message is tagged with a millisecond accurate realtime clock stamp, as well as a PCM Count timestamp. The Realtime clock is useful for comparing transmit to receive times between units, and the PCM clock gives an indication of when a message is displayed with respect to reception or transmission of acoustic messages.
- The Logfile is Rolling: Each time the Popoto app is started, the previous log file is added to a list of 10 preceeding log files. So that in the Popoto_app directory we have Popoto.log, Popoto.log.1, Popoto.log.2 through Popoto.log.10 where Popoto.log is the current logfile, and Popoto.log.1 is the most recent log file preceding this logfile.

20.1.2 Location

On the target hardware the Popoto.log file is found in the /home/root directory. On the PC-Based Linux simulation, the Popoto.log is found in the /tmp directory. In order to allow more than one Popoto image to run on a pc, the base-port number is appended to the Popoto.log filename. /tmp/Popoto.log.17000 Corresponds to a Popoto image run at a base port of

17000 Or /tmp/Popoto.log.18000 Corresponds to a Popoto image run at a base port of 18000. For example:

20.1.3 Logging Levels

Each log message is assigned a logging level from 0 to 7. Lower log levels are more severe, and higher log levels are increasing details. follows:

0. logERROR
1. logWARNING
2. logINFO
3. logDEBUG
4. logDEBUG1
5. logDEBUG2
6. logDEBUG3
7. logDEBUG4

The log levels are defined as By default all log messages with a logging level of logINFO or lower are written to the log. To increase or decrease the log level issue the `SetValue LoggingLevel int <Level> 0` command Or from the pshell:

```
setvaluei LoggingLevel <level>
```

To get the current logging level, issue the `GetValue LoggingLevel int 0` command, Or from the pshell:

```
getvaluei LoggingLevel
```

20.1.4 MSM Logs

The Modem State Machine has a built in logging mechanism that can be connected to the Popoto.log file. This allows the user to see events, and state transitions as the modem state machine operates. To enable the MSM logs, send the command `EnableMSMLogs`. Or, from the pshell: `enablemsmlogs`

To disable logs, send the `DisableMSMLogs` command. Or from the pshell:

```
disablemsmlogs
```

20.2 PCM Logging

20.2.1 Introduction

The Popoto system incorporates a means for logging the inbound PCM signals as seen on the A/D. This logging mechanism is useful for diagnosing system problems. Since the

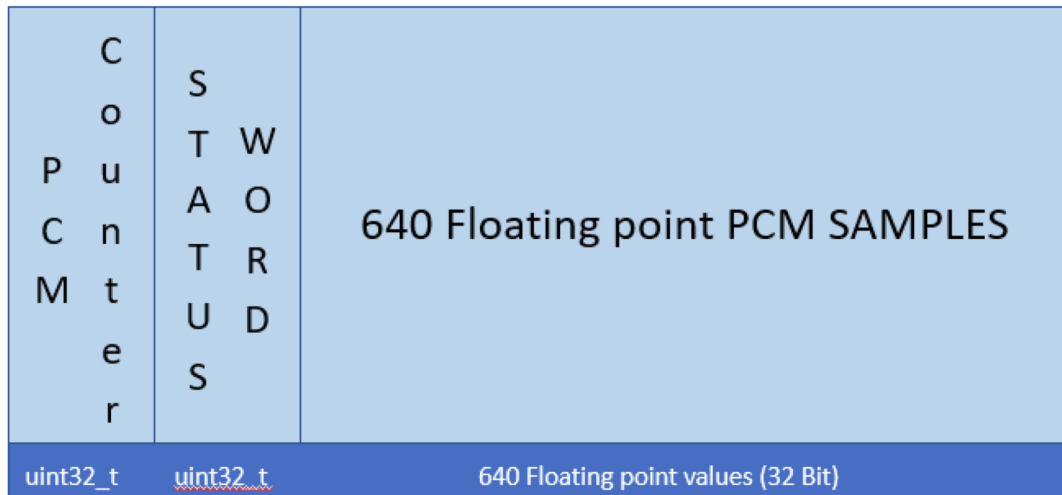


Figure 20.1: Format of a single PCM Log Packet. These packets are transmitted on the TCP PCM Recording socket.

PCM signals that are logged are exactly what is presented to the Demodulator, it possible to “re-run” a test condition, to determine the signal parameters or noise environment. Two methods of logging are provided to the user:

1. TCP Socket Based Logging
2. Target File Logging

Each of these methods produces a data stream of packets that are formatted as follows:

Table 20.1: PCM Packet Format

Count	Data type	Description
1	32 Bit unsigned int	PCM Counter. Gives the current PCM counter. Should increase by 640 each frame. A skip in this count indicates lost data
1	32 Bit unsigned int	Status Word. Currently 0 indicates High Gain Channel, and 1 indicates low gain channel
640	32 Bit Floating point	PCM Samples. All normalized to High Gain Receive Level.

20.2.2 Socket based PCMLogs

The Popoto system opens a TCP Server at baseport+2 (17002 default) which continually streams PCM Packets as described above. Both the Popoto.py



Figure 20.2: The PCM Packets are sent one after the other to the TCP Socket or to the Target log file

and Popoto.m interface classes have functions to read that socket and log the data to the local pc.

From the pshell

```
recordstart <Filename> local
```

will start the recording in the current working directory. To stop the recording: From the pshell:

```
recordstop
```

20.2.3 Target File based PCM Logs

The Popoto system provides a command to store the received pcm locally. Using the

```
RecordFileStart <FileName>
```

command, the user can start logging data to the local SD card.

If the filename is specified without a path, it will be recorded in `/home/root`. paths should be complete paths. Wild cards are not parsed.

From the pshell:

```
recordstart WaterTestCapeCodCana12_20.pcm
```

will begin a recording on the Popoto unit in the `/home/root` directory

To stop the recording, a RecordFileStop command can be sent on the command socket. Or, from the pshell:

```
recordstop
```

A Matlab utility: `rPCMData()` is provided in the `test/MATLAB GUI` directory. This utility can read a file logged by the pshell or by the Target recording, and returns 3 arrays, the PCM data, the PCM Counter(sequence number) and the status word.

20.2.4 Notes

It is important to realize that PCM recording generates data very quickly. Each packet is

$$642 \times 4$$

Bytes long, and 160 packets are generated per second. This results in a file that grows at 410,810 bytes per second, or roughly 1.5 G Bytes per hour.

20.3 pshell Logging

The pshell provides a log of all commands and status responses for a pshell session. This is useful for capturing the results of tests, or to evaluate the responses and commands that were run. pshell logs are size-limited, and rotate. These logs can be found in the directory that the pshell was run in.

21 Automatic PCM Error Logging

The Popoto Application includes a diagnostic feature called **Automatic PCM Error Logging**, designed to support offline analysis of signal quality during transmission failures.

21.1 Overview

When this feature is enabled, the Popoto App continuously maintains a rolling buffer of the most recent 30 seconds of PCM passband data. If a transmission fails with either a **CRC Error** or a **Ranging Error**, the application will automatically dump this buffered audio into a timestamped '.wav' file.

These recordings are intended for developers and engineers to use in post-mission analysis, regression testing, or reproducing field issues under known acoustic conditions.

21.2 How to Enable

This feature is controlled at runtime via the 'pshell' interface.

- `set LogPcmEnable 0` Disables PCM error logging (default)
- `set LogPcmEnable 1` Enables automatic logging of recent PCM data upon error

No additional configuration is required. When enabled, the Popoto App will write '.wav' files automatically when a relevant error is detected.

21.3 Logging Details

- The recorded files are mono standard WAV format at the application sample rate and timestamped at the moment of error detection.
- Only raw PCM passband audio is captured — decoded messages or payload content are not included.

- Logging is triggered specifically by **CRC Errors** and **Ranging Errors**.
- The PCM buffer is always 30 seconds long.

22 Pshell Command Reference

pshell Command: Rx

Description:

Rx Receive packets and format the output for test purposes. Continues to run until a key is hit.

Invocation

Rx [Verbose Flag]

Verbose Flag = 1 Output SNR and Doppler info

Examples

Rx

Enter test receive in quiet mode

Rx 1

Enter test receive in verbose mode.

pshell Command: chat

Description:

This command puts Popoto into a character chat mode, In chat mode, the user can type characters, and they will be transmitted when one of 2 conditions occur. 1) the user stops typing for a period of time greater than ConsoleTimeoutMS, or 2) a string of characters greater in length than ConsolePacketBytes is typed. ConsoleTimeoutMS and ConsolePacketBytes are Settable Variable parameters.

Invocation

```
chat
```

Examples

```
chat  
ctrl-] to exit
```

pshell Command: configure

Description:

This api configures the modem for different modulation schemes. It is used to allow switching between major operating modes such as Janus and default Popoto modes. Invocation of this command issues a reboot, after which the modem is in the new mode of operation.

Invocation

```
configure <MODE>
```

Examples

```
configure Janus  
to setup Janus mode  
configure Popoto  
to setup Popoto Mode
```

pshell Command: connect

Description:

The connect command is used to connect the pshell with the command socket. This is typically the first command executed in the session of a pshell. A successful connection responds with the list of available parameters.

Invocation

```
connect <ipaddress> <port>
```

Examples

```
connect localhost 17000  
connect 10.0.0.232 17000
```

pshell Command: datamode

Description:

This command ends voice mode, and returns the device to data mode,

Invocation

```
datamode
```

Examples

```
datamode
```

pshell Command: deepsleep

Description:

Place Popoto into Deep Sleep mode to be awakened by a wake up tone on the acoustic interface. Once in deep sleep, any 25Khz acquisition pattern will wake the popoto modem. This can most easily be generated by sending a ping command from the remote modem. Deepsleep is a low power mode that consumes 150mW. Awakening from Deepsleep takes approximately 1 second after the acquisition.

Invocation

```
deepsleep
```

Examples

```
deepsleep
```

pshell Command: disablemsmlog

Description:

This api disables logging of modem statemachine transitions.

Invocation

```
disablemsmlog
```

Examples

```
disablemsmlog
```

pshell Command: disconnect

Description:

Disconnect the pshell from the popoto modem application. This command is sent if the user wishes to connect an application via ethernet.

Invocation

```
disconnect
```

Examples

```
disconnect
```

pshell Command: download

Description:

downloads a file in streaming mode. The remote unit must issue an upload. if the start remote start power level is set to other than 0, the local modem will send an upload command to the remote modem using the specified power level., and then begin the download process. Otherwise it will sit and wait for the remote modem to start on its own.

Invocation

```
download <filename> [Remote Start Power Level]
```

Examples

```
download MyDownload.txt  
download MyDownload.txt 10
```

pshell Command: enablemsmlog

Description:

This api enable logging of modem statemachine transitions. These transition sare logged in the popoto.log file on the modem, and are noted with the ENTER STATE text

Invocation

```
enablemsmlog
```

Examples

```
enablemsmlog
```

pshell Command: exit

Description:

Exits Popoto Modem pshell. Note: On hardware pshell, quit and exit are disabled

Invocation

```
exit
```

Examples

```
exit
```

pshell Command: getEXP1

Description:

The EXP1 Pin is a GPIO Input pin available on the Popoto expansion header. This API allows the user to get the value of that pin.

Invocation

```
getEXP1
```

Examples

```
getEXP0
```

pshell Command: getIP

Description:

Display the currently configured IP address and status of the Popoto modem

Invocation

```
getIP
```

Examples

```
getIP
```

```
IPv4 Address: eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  
mtu 1500  
inet 10.0.0.65 netmask 255.255.255.0 broadcast 10.0.0.255  
ether 00:0c:29:36:4f:2f txqueuelen 1000 (Ethernet)  
RX packets 3178079 bytes 843820500 (843.8 MB)  
RX errors 0 dropped 508 overruns 0 frame 0  
TX packets 2392420 bytes 2432926671 (2.4 GB)  
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

pshell Command: getPEP

Description:

Returns the peak envelope power of the transmitted waveform. PEP is a metric used to quantify the voice transmit power.

Invocation

```
getPEP
```

Examples

```
getPEP
```

pshell Command: getclock

Description:

Get the Realtime clock in the format YYYY.MM.DD-HH:MM;SS

Invocation

```
getclock
```

Examples

```
getclock
```

```
2021.04.02-10:22:30
```

```
get the Realtime clock in the format YYYY.MM.DD-HH:MM;SS
```

pshell Command: `getvaluef`

Description:

(DEPRECATED) Returns the value of an floating point variable within the Popoto modem. This API is deprecated in favor of the simpler pshell api which allows getting variables without a command. See examples below.

Invocation

```
getvaluef <Element>
```

Examples

```
getvaluef TxPowerWatts
```

This expression can be replaced with the simpler

```
TxPowerWatts
```

Both will return a JSON message like:

```
{"TxPowerWatts":1.000000}
```

pshell Command: getvaluei

Description:

(DEPRECATED) Returns the value of an integer variable within the Popoto modem. This API is deprecated in favor of the simpler pshell api which allows getting variables without a command. See examples below.

Invocation

```
getvaluei <Element>
```

Examples

```
getvaluei UPCONVERT_Carrier
```

This expression can be replaced with the simpler

```
UPCONVERT_Carrier
```

Both will return a JSON message like:

```
{"UPCONVERT_Carrier":25000}
```

pshell Command: getverbosity

Description:

The `getverbosity` command is used to read the current verbosity of the popoto api This command returns an integer from 0 to 5. 0 = silent 5 = most verbose

Invocation

```
getverbosity
```

Examples

```
getverbosity
```

pshell Command: ls

Description:

ls generates a directory listing of the local Popoto storage. it takes 2 arguments. 1) a directory name 2) a regular expression to match for the files to list.

Invocation

```
connect <ipaddress> <port>
```

Examples

```
connect localhost 17000
connect 10.0.0.232 17000
print a directory listing
ls <directory name> <regex>
ls /captures
ls . *.rec
```

pshell Command: mips

Description:

Query the popoto modem to determine internal cycle counts for algorithms. Cycle counts are returned in a JSON dictionary for parsing by Popoto development tools. This is a typically a command used by the developers.

Invocation

```
mips
```

Examples

```
mips
```

pshell Command: multiping

Description:

Send an series of acoustic test messages. This api sends the text "Popoto Test Message" repeatedly using the configured data rate, and the approximate specified power level. This api is used to run packet level reliability checks. The power is specified, along with a count, and an interpacket delay.

Invocation

```
multiping <power Watts> <number of pings> <delay in seconds>
```

Examples

```
multiping 10 20 5
```

Will send 20 ping messages at 10 watts with 5 seconds of delay between messages

pshell Command: netplay

Description:

Plays a file file using the network sockets

Invocation

```
netplay <delresearchfile> <scale> <BB/PB>
```

where

delresearchfile: is a valid filename

scale: is a floating point gain to be applied to the signal p
prior to transmission

BB/PB: Baseband or passband 1 -> Baseband Recording 0->Passband
Recording

Base band carrier is selected by setting the BBAND_PBAND_UpCarrier
variable.

Examples

```
netplay TestPBRecording 1.0 0
```

plays the file TestPBRecording for at a gain of 1.0 in Passband

```
netrec TestBBRecording 20 1
```

records the file TestBBRecording at a gain of 1.0 in Baseband

pshell Command: netrec

Description:

Records a file file using the network sockets

Invocation

```
netrec <delresearch File> <time in seconds> <BB/PB>
```

where

delresearch file is a valid filename

time in seconds is the desired length of the recording

BB/PB=1 -> Baseband Recording 0->Passband Recording

Base band carrier is selected by setting the BBAND_PBAND_DownCarrier variable.

Examples

```
netrec TestRecording 20 0
```

records the file TestRecording for 20 seconds in Passband

```
netrec TestRecording 20 1
```

records the file TestRecording for 20 seconds in Baseband

pshell Command: ping

Description:

Send an acoustic test message. This api sends the text "Popoto Test Message" using the configured data rate, and the approximate specified power level. It is important to note that calling ping with a power level latches that power level in the transmitter, to be used for subsequent transmissions.

Invocation

```
ping <Power level>
```

Examples

```
ping 10
```

Sends a test message (Popoto Test Message) using approximately 10 watts of power

pshell Command: playstart

Description:

Starts a playback from the local modem's filesystem. where filename is the name of the file to play where scale factor is a floating point gain to apply to the file

Invocation

```
playstart <filename> <scale factor>
```

Examples

```
playstart /captures/Tone.pcm 1.0
```

pshell Command: playstop

Description:

Stop and close an in-process playback

Invocation

```
playstop
```

Examples

```
playstop
```

pshell Command: powerdown

Description:

Place Popoto into POWERDOWN mode to be awakened by a wake up tone on the acoustic interface. Once in powerdown mode, any 25Khz acquisition pattern will wake the popoto modem. This can most easily be generated by sending a ping command from the remote modem. Things to note: Powerdown mode is the lowest power state of the Popoto Modem, typically 13mW. To awaken from Powerdown mode requires 20 seconds after the acquisition.

Invocation

```
deepsleep
```

Examples

```
deepsleep
```

pshell Command: q

Description:

Minimize (quiet) the output to the console during normal operation.

Invocation

q

Examples

q

pshell Command: quit

Description:

An alias for exit. Exits Popoto Modem pshell. Note: On hardware pshell, quit and exit are disabled

Invocation

```
quit
```

Examples

```
quit
```

pshell Command: range

Description:

Sends a two way range request using approximately <Power> watts. This command issues a range request and sends it to the modem at the configured remoteID. The remote modem holds the request for a predetermined amount of time, and then replies with a range response. Popoto will then send back a range report consisting of the distance between the modems, and the configured speed of sound and the computed round trip time. Note that the Speed of sound, and the ranging hold time are configurable parameters, if you do change the ranging hold time, it is imperative that you configure both the local and remote modems to have the same hold time. Otherwise, Popoto will give erroneous range reports.

Invocation

```
range <power>
```

Examples

```
range 20
{"Range":500.002441,"Roundtrip Delay":666.669922,"SpeedOfSound":1500.000000,"Units":"m,
ms, meters per second"}
```

pshell Command: recordstart

Description:

starts a recording to the local storage device.. Filenames are extended with a timestamp. The file(s) will continue to record until the recordstop command is issued

Invocation

```
recordstart <filename> [duration]
```

where

filename: is the name of the file to record on the target processor

duration: Optional parameter that tells how long each individual record file length is in seconds.

Examples

```
recordstart /captures/TestCapeCodBay 60
```

records a file called TestCapeCodBay<Timestamp>.rec, and rolls the file every 60 seconds, starting a new file with the same base filename with a new appended timestamp

pshell Command: recordstop

Description:

Stop and close an in-process recording

Invocation

```
recordstop
```

Examples

```
recordstop
```

pshell Command: remote

Description:

Toggles remote mode. In remote mode, any command issued at the pshell is wrapped into an acoustic message and transmitted to the remote modem, where the command is executed, and the status is returned in an acoustic message from the remote modem. Note: It is not permissible to issue a remote transmission using remote mode.

Invocation

```
remote <on/off>
```

Examples

```
remote on
```

Enables remote mode

```
remote off
```

Disables remote mode

NOTE: You cannot issue a transmit command remotely

pshell Command: setEXP0

Description:

The EXP0 Pin is a GPIO Output pin available on the Popoto expansion header. This API allows the user to set the value of that pin. Note that the GPIO pin has limited current drive, and if a high current device is to be controlled, it is necessary to use an external FET or relay. Please see Popoto.com for application notes concerning controlling high current devices.

Invocation

```
setEXP0 <1,0>
```

Examples

```
setEXP0 0  
Turn off the EXP0 pin  
setEXP0 1  
Turn on the EXP0 pin
```

pshell Command: setRate10240

Description:

Set the modem payload transmission rate to 10240 bits per second

Invocation

```
do_setRate10240
```

Examples

```
do_setRate10240
```

NOTE: This modulation rate is UNCODED, and will only work on very clean channels

Use with caution.

pshell Command: setRate1280

Description:

Set the modem payload transmission rate to 1280 bits per second

Invocation

```
setRate1280
```

Examples

```
setRate1280
```

```
Set the local modem to use the 1280 bit per second modulation scheme
```

pshell Command: setRate2560

Description:

Set the modem payload transmission rate to 2560 bits per second

Invocation

```
setRate2560
```

Examples

```
setRate2560
```

pshell Command: setRate5120

Description:

Set the modem payload transmission rate to 5120 bits per second

Invocation

```
setRate5120
```

Examples

```
setRate5120
```

pshell Command: setRate640

Description:

Set the modem payload transmission rate to 640 bits per second

Invocation

```
setRate640
```

Examples

```
setRate640
```

Set the local modem to use the 640 bit per second modulation scheme

pshell Command: setRate80

Description:

Set the modem payload transmission rate to 80 bits per second

Invocation

```
setRate80
```

Examples

```
setRate80
```

pshell Command: setTerminalMode

Description:

Set the pshell terminal to raw mode or ANSI mode. ANSI Mode allows for highlighting of responses, Raw mode is easier to use if controlling the device programatically

Invocation

```
setTerminalMode <raw/ansi>
```

Examples

```
setTerminalMode raw  
setTerminalMode ansi
```

pshell Command: setcarrier

Description:

A helper function to set the transmit and receive carriers to a value. Note that given the version of the modem, there will be different bounds for carrier frequencies. Check documentation UPCONVERT_Carrier and DOWNCONVERT_Carrier for details on acceptable ranges.

Invocation

```
setcarrier <Carrier Frequency>
```

Examples

```
setcarrier 25000
```

pshell Command: setcarrier25

Description:

A helper function to set the transmit and receive carriers to 25Khz

Invocation

```
setcarrier25
```

Examples

```
setcarrier25
```

pshell Command: setcarrier30

Description:

A helper function to set the transmit and receive carriers to 30Khz

Invocation

```
setcarrier30
```

Examples

```
setcarrier30
```

pshell Command: setclock

Description:

Set the Realtime clock in the format YYYY.MM.DD-HH:MM;SS

Invocation

```
setclock <Date Time>
```

Examples

```
setclock 2021.04.02-10:22:30
```

pshell Command: setgainmode

Description:

Sets the way the modem manages the high and low gain channels

Invocation

```
setGainMode <0,1,2>  
GainMode 0 = High Gain Only  
GainMode 1 = Low Gain Only  
GainMode 2 = Automatic Gain Selection
```

Examples

```
setGainMode 2
```

pshell Command: setvaluef

Description:

(DEPRECATED) Sets an floating point value on the popoto modem This API is deprecated in favor of the simpler pshell api which allows setting variables without a command. See examples below.

Invocation

```
setvaluef <Element>
```

Examples

```
setvaluef TxPowerWatts 10.0
```

This expression can be replaced with the simpler

```
TxPowerWatts 10.0
```

pshell Command: setvaluei

Description:

(DEPRECATED) Sets an integer value on the popoto modem This API is deprecated in favor of the simpler pshell api which allows setting variables without a command. See examples below.

Invocation

```
setvaluei <Element>
```

Examples

```
setvaluei UPCONVERT_Carrier 30000
```

This expression can be replaced with the simpler
`UPCONVERT_Carrier 30000`

pshell Command: setverbosity

Description:

The setverbosity command is used to control the verbosity of the popoto api This command takes an integer from 0 to 5. 0 = silent 5 = most verbose

Invocation

```
setverbosity <value>
```

Examples

```
setverbosity 0  
setverbosity 2
```

pshell Command: sleep

Description:

This command pauses the pshell for N Seconds. It is useful when writing scripts or commands that need to perform tasks at a prescribed interval.

Invocation

```
sleep <N>  
Sleep for N seconds, where N is an integer.
```

Examples

```
sleep 5
```

pshell Command: ssb

Description:

Place the ssb Voice into Receive mode

Invocation

```
ssb
```

Examples

```
ssb
```

pshell Command: ssbtx

Description:

Force the SSB Voice mode into Transmit mode

Invocation

```
ssbtx
```

Examples

```
ssbtx
```

pshell Command: startrx

Description:

This command enables the modem receiver, and returns the modem state machine to the listening state pshell invokes this command automatically at boot up.

Invocation

```
startrx
```

Examples

```
startrx
```

pshell Command: transmit

Description:

Transmit a string to the remote modem. Strings do not need to be delimited, and can have spaces in them. This is used for sending data to the remote modem

Invocation

```
transmit <message>
```

Where message is a text string

Examples

```
transmit Hello  
transmit Hello World it's me, Popoto
```

pshell Command: transmitJSON

Description:

Transmit a JSON encoded message to the remote modem. This is used for sending data to the remote modem

Invocation

```
transmitJSON <message>
```

The structure of the message is

```
{"Payload":{"Data": [<COMMA SEPARATED 8 BIT VALUES>]}}
```

Examples

```
transmitJSON {"Payload":{"Data": [1,2,3,4,5]}}
```

sends the binary sequence 0x01 0x02 0x03 0x04 0x05

```
transmitJSON {"Payload":{"Data": "Hello World"}}
```

sends the text sequence Hello World

pshell Command: transmitJSONFiles

Description:

Transmit a file of JSON encoded messages to the remote modem.

Invocation

```
transmitJSONFiles <filename> <power> <delay between transmissions>  
<num transmissions per packet>
```

Examples

```
transmitJSONFiles JanusTestCase1.txt 10 30 10
```

pshell Command: unq

Description:

Unquiet the output to the console during normal operation.

Invocation

```
unq
```

Examples

```
unq
```

pshell Command: upload

Description:

Uploads a file in streaming mode.

Invocation

```
upload [filename] [power level]
```

Examples

```
upload myfile 10
```

pshell Command: version

Description:

Return the serial number and software version of the Popoto modem. Each item is returned in an informational JSON message as shown below

Invocation

```
version
```

Examples

```
version
```

```
{"Info ":"Popoto Modem Version 2.7.0 847"}  
{"Info ":"SerialNumber FFFFFFFFFFFFFFFFFFFFFF"}
```

23 Popoto Variables

The Popoto modem system has a database of configurable parameters which allow customization of the operation of the Popoto modem. These parameters, referred to as Settable/Gettable Variables provide system information such as battery voltage, control modulation parameters such as transmission power and carrier frequency, and provide runtime status such as constellation points, and PLL errors. Settable/Gettable Variables have permissions and bounds checking associated with them. It is important to note that some variables, such as BatteryVoltage, are read-only, and some variables such as UPCONVERT_Carrier, are read and writeable.

Setting variables is accomplished within the JSON API using the Set command. In the example below, we set the Baseband Recording downconvert carrier to 45Khz. The format of the message is:

```
{"Command": "Set", "Arguments": "SPACE DELIMITED ARGUMENT LIST"}
```

Table 23.1: Argument List format

Variable	Data Type	Value	Channel
BBAND_DownCarrier	int	Value	0

```
{"Command": "Set", "Arguments": "BBAND_DownCarrier int 45000.0 0" }
```

Table 23.2: Variable Set Return Conditions

API	Condition	Example Return
{"Command": "SetValue", "Arguments": "BBAND_DownCarrier 25000 0"}	Success	{"Info": "Value Set BBAND_DownCarrier=25000"}
{"Command": "SetValue", "Arguments": "BBAND_DownCarrier 5000 0"}	Below Min	{"Info": "Value Out of Range: BBAND_DownCarrier=5000 Below Minimum"}
{"Command": "SetValue", "Arguments": "BBAND_DownCarrier 500000 0"}	Above Max	{"Info": "Value Out of Range: BBAND_DownCarrier=500000 Above Maximum"}
{"Command": "SetValue", "Arguments": "bband_downcarrier 25000 0"}	Misspelled	{"Error": "Unknown Element bband_downcarrier"}

What follows is a reference for all of the controllable variables within the Popoto Modem system

APP_CycleCount

Description:

Display Cycle Counter

Data Type

int

Minimum Value

0.0

Maximum Value

0.0

Permissions

Read Only

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "APP_CycleCount int 0 }
```

```
{"Command": "SetValue", "Arguments": "APP_CycleCount int 0.0 0 }
```

Return:

```
{"APP_CycleCount": value}
```

APP_CycleCountReset

Description:

Display Cycle Counter and Reset

Data Type

int

Minimum Value

0.0

Maximum Value

0.0

Permissions

Read Only

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "APP_CycleCountReset int 0 }
```

```
{"Command": "SetValue", "Arguments": "APP_CycleCountReset int 0.0 0 }
```

Return:

```
{"APP_CycleCountReset": value}
```

APP_ModemSMAOut

Description:

Flag to Send Modem Data to the SMA Port out

Data Type

int

Minimum Value

0.0

Maximum Value

1.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "APP_ModemSMAOut int 0 }
```

```
{"Command": "SetValue", "Arguments": "APP_ModemSMAOut int 1.0 0 }
```

Return:

```
{"APP_ModemSMAOut": value}
```

APP_SocketBasedPCM

Description:

Flag to enable Socket based PCM

Data Type

int

Minimum Value

0.0

Maximum Value

1.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "APP_SocketBasedPCM int 0 }
```

```
{"Command": "SetValue", "Arguments": "APP_SocketBasedPCM int 1.0 0 }
```

Return:

```
{"APP_SocketBasedPCM": value}
```

BBAND_DownCarrier

Description:

Downconvert Baseband Streaming carrier 5120 to 45000

Data Type

int

Minimum Value

5120.0

Maximum Value

45000.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "BBAND_DownCarrier int 0" }  
{"Command": "SetValue", "Arguments": "BBAND_DownCarrier int 45000.0 0" }  
}
```

Return:

```
{"BBAND_DownCarrier": value}
```

BBAND_OutputScale

Description:

Upconvert Output scaling for Baseband Passband module

Data Type

float

Minimum Value

0.0

Maximum Value

10.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "BBAND_OutputScale float 0 }
```

```
{"Command": "SetValue", "Arguments": "BBAND_OutputScale float 10.0 0 }
```

Return:

```
{"BBAND_OutputScale": value}
```

BBAND_UpCarrier

Description:

Upconvert Baseband Streaming carrier 5120 to 45000

Data Type

int

Minimum Value

5120.0

Maximum Value

45000.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "BBAND_UpCarrier int 0 }
```

```
{"Command": "SetValue", "Arguments": "BBAND_UpCarrier int 45000.0 0 }
```

Return:

```
{"BBAND_UpCarrier": value}
```

BatteryCharge

Description:

Battery Charge in mAh

Data Type

float

Minimum Value

0.0

Maximum Value

271.98999

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "BatteryCharge float 0 "}
{"Command": "SetValue", "Arguments": "BatteryCharge float 271.98999 0 "}
}
```

Return:

```
{"BatteryCharge": value}
```

BatteryCurrent

Description:

Battery Current in positive (to battery; unused) / negative (from battery) in mA

Data Type

float

Minimum Value

0.0

Maximum Value

200.0

Permissions

Read Only

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "BatteryCurrent float 0" }
```

```
{"Command": "SetValue", "Arguments": "BatteryCurrent float 200.0 0" }
```

Return:

```
{"BatteryCurrent": value}
```

BatteryVoltage

Description:

System Battery Voltage in volts

Data Type

float

Minimum Value

0.0

Maximum Value

40.0

Permissions

Read Only

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "BatteryVoltage float 0 }
```

```
{"Command": "SetValue", "Arguments": "BatteryVoltage float 40.0 0 }
```

Return:

```
{"BatteryVoltage": value}
```

Carrier

Description:

Sets both the Transmit and Receiver Carriers. Reading this variable returns the Transmitter Carrier.

Data Type

int

Minimum Value

20000.0

Maximum Value

59750.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "Carrier int 0" }
```

```
{"Command": "SetValue", "Arguments": "Carrier int 59750.0 0" }
```

Return:

```
{"Carrier": value}
```

CarrierTxMode

Description:

set the transmitter to send FH waveform (0-default) or 1-simply a carrier note

Data Type

int

Minimum Value

0.0

Maximum Value

1.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "CarrierTxMode int 0" }
```

```
{"Command": "SetValue", "Arguments": "CarrierTxMode int 1.0 0" }
```

Return:

```
{"CarrierTxMode": value}
```

ChannelIR

Description:

Measured channel impulse response: Getter returns number of expected values

Data Type

int

Minimum Value

0.0

Maximum Value

0.0

Permissions

Read Only

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "ChannelIR int 0" }  
{"Command": "SetValue", "Arguments": "ChannelIR int 0.0 0" }
```

Return:

```
{"ChannelIR": value}
```

ConsolePacketBytes

Description:

Number of console characters input to trigger an autosend

Data Type

int

Minimum Value

0.0

Maximum Value

8192.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "ConsolePacketBytes int 0" }
```

```
{"Command": "SetValue", "Arguments": "ConsolePacketBytes int 8192.0 0" }
```

```
}
```

Return:

```
{"ConsolePacketBytes": value}
```

ConsoleTimeoutMS

Description:

Number of console milliseconds expired to trigger an autosend

Data Type

int

Minimum Value

0.0

Maximum Value

60000.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "ConsoleTimeoutMS int 0 }
```

```
{"Command": "SetValue", "Arguments": "ConsoleTimeoutMS int 60000.0 0 }
```

Return:

```
{"ConsoleTimeoutMS": value}
```

DOWNCONVERT_Carrier

Description:

Downconverter Carrier Frequency in Hz

Data Type

int

Minimum Value

20000.0

Maximum Value

59750.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "DOWNCONVERT_Carrier int 0 }
```

```
{"Command": "SetValue", "Arguments": "DOWNCONVERT_Carrier int 59750.0  
0 }
```

Return:

```
{"DOWNCONVERT_Carrier": value}
```

DataPortMode

Description:

0-Data Port acts as Telnet; 1 Data Port is raw TCP data

Data Type

int

Minimum Value

0.0

Maximum Value

1.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "DataPortMode int 0" }
```

```
{"Command": "SetValue", "Arguments": "DataPortMode int 1.0 0" }
```

Return:

```
{"DataPortMode": value}
```

DepartureDelay_samples

Description:

Tx departure delay

Data Type

int

Minimum Value

0.0

Maximum Value

3072000.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "DepartureDelay_samples int 0 }
```

```
{"Command": "SetValue", "Arguments": "DepartureDelay_samples int
```

```
3072000.0 0 }
```

Return:

```
{"DepartureDelay_samples": value}
```

DopplerDisable

Description:

Switch to control the doppler compensation 1= Doppler Compensation on 0 = Doppler Compensation off

Data Type

int

Minimum Value

0.0

Maximum Value

1.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "DopplerDisable int 0 }
```

```
{"Command": "SetValue", "Arguments": "DopplerDisable int 1.0 0 }
```

Return:

```
{"DopplerDisable": value}
```

DopplerEnable

Description:

enable (1) or disable(0) doppler estimation

Data Type

int

Minimum Value

0.0

Maximum Value

1.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "DopplerEnable int 0 }
```

```
{"Command": "SetValue", "Arguments": "DopplerEnable int 1.0 0 }
```

Return:

```
{"DopplerEnable": value}
```

FHDEMODO_DetectThresholdDB

Description:

Detection threshold for signal acquire default 160 for -5db AWGN detect

Data Type

float

Minimum Value

0.0

Maximum Value

300.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "FHDEMODO_DetectThresholdDB float 0" }
```

```
{"Command": "SetValue", "Arguments": "FHDEMODO_DetectThresholdDB float 300.0 0" }
```

Return:

```
{"FHDEMODO_DetectThresholdDB": value}
```

FHDEMODO_HopSeqLen

Description:

Set the hop sequence length demodulator

Data Type

int

Minimum Value

1.0

Maximum Value

8192.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "FHDEMODO_HopSeqLen int 0 "}
{"Command": "SetValue", "Arguments": "FHDEMODO_HopSeqLen int 8192.0 0 }
```

Return:

```
{"FHDEMODO_HopSeqLen": value}
```

FHMOD_HopSeqLen

Description:

Set the hop sequence length demodulator

Data Type

int

Minimum Value

1.0

Maximum Value

8192.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "FHMOD_HopSeqLen int 0 }
```

```
{"Command": "SetValue", "Arguments": "FHMOD_HopSeqLen int 8192.0 0 }
```

Return:

```
{"FHMOD_HopSeqLen": value}
```

FHMOD_NumChipReps

Description:

Set the hop frequency redundancy for payloads

Data Type

int

Minimum Value

1.0

Maximum Value

4.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "FHMOD_NumChipReps int 0 }
```

```
{"Command": "SetValue", "Arguments": "FHMOD_NumChipReps int 4.0 0 }
```

Return:

```
{"FHMOD_NumChipReps": value}
```

GainAdjustMode

Description:

Set the gain mode 0-lowgain, 1-highgain, 2-automatic

Data Type

int

Minimum Value

0.0

Maximum Value

2.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "GainAdjustMode int 0 }
```

```
{"Command": "SetValue", "Arguments": "GainAdjustMode int 2.0 0 }
```

Return:

```
{"GainAdjustMode": value}
```

HP_FilterEnable

Description:

HP Filter enable / disable flag 0=disable 1 = enable

Data Type

int

Minimum Value

0.0

Maximum Value

1.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "HP_FilterEnable int 0 }
```

```
{"Command": "SetValue", "Arguments": "HP_FilterEnable int 1.0 0 }
```

Return:

```
{"HP_FilterEnable": value}
```

Headerless

Description:

Enable headerless streaming mode (0-default)

Data Type

int

Minimum Value

0.0

Maximum Value

1.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "Headerless int 0" }
```

```
{"Command": "SetValue", "Arguments": "Headerless int 1.0 0" }
```

Return:

```
{"Headerless": value}
```

InBandNoiseEnergy

Description:

Noise Energy Measured after downsampling filter

Data Type

float

Minimum Value

0.0

Maximum Value

1.0

Permissions

Read Only

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "InBandNoiseEnergy float 0 }
```

```
{"Command": "SetValue", "Arguments": "InBandNoiseEnergy float 1.0 0 }
```

Return:

```
{"InBandNoiseEnergy": value}
```

InbandEnergy

Description:

Inband energy parameter

Data Type

float

Minimum Value

0.0

Maximum Value

0.0

Permissions

Read Only

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "InbandEnergy float 0 }
```

```
{"Command": "SetValue", "Arguments": "InbandEnergy float 0.0 0 }
```

Return:

```
{"InbandEnergy": value}
```

LedEnable

Description:

0-disable all board LEDS; 1 enable board LEDS

Data Type

int

Minimum Value

0.0

Maximum Value

1.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "LedEnable int 0" }
```

```
{"Command": "SetValue", "Arguments": "LedEnable int 1.0 0" }
```

Return:

```
{"LedEnable": value}
```

LocalID

Description:

Local Modem ID 0-254;255 broadcast

Data Type

int

Minimum Value

0.0

Maximum Value

255.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "LocalID int 0" }
```

```
{"Command": "SetValue", "Arguments": "LocalID int 255.0 0" }
```

Return:

```
{"LocalID": value}
```

LoggingLevel

Description:

Logging verbosity level

Data Type

int

Minimum Value

0.0

Maximum Value

5.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "LoggingLevel int 0 }
```

```
{"Command": "SetValue", "Arguments": "LoggingLevel int 5.0 0 }
```

Return:

```
{"LoggingLevel": value}
```

MODEM_Enable

Description:

enable (1) or disable (0) modem processing

Data Type

int

Minimum Value

0.0

Maximum Value

1.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "MODEM_Enable int 0 }
```

```
{"Command": "SetValue", "Arguments": "MODEM_Enable int 1.0 0 }
```

Return:

```
{"MODEM_Enable": value}
```

PSK_BnTaps

Description:

The number of Backward taps for the PSK Equalizer. The number of forward taps + the number of backwards taps must be less than MAX value

Data Type

int

Minimum Value

0.0

Maximum Value

100.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "PSK_BnTaps int 0 "}
{"Command": "SetValue", "Arguments": "PSK_BnTaps int 100.0 0 "}
```

Return:

```
{"PSK_BnTaps": value}
```

PSK_Constellation

Description:

Returns the last 64 Constellation points from the PSK Modem

Data Type

float

Minimum Value

0.0

Maximum Value

0.0

Permissions

Read Only

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "PSK_Constellation float 0 }
```

```
{"Command": "SetValue", "Arguments": "PSK_Constellation float 0.0 0 }
```

Return:

```
{"PSK_Constellation": value}
```

PSK_FnTaps

Description:

The number of Forward taps for the PSK Fractional (N/2) Equalizer. The number of forward taps + the number of backwards taps must be less than MAX value.

Data Type

int

Minimum Value

0.0

Maximum Value

100.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "PSK_FnTaps int 0 "}
{"Command": "SetValue", "Arguments": "PSK_FnTaps int 100.0 0 "}
```

Return:

```
{"PSK_FnTaps": value}
```

PSK_PDSNR

Description:

Post detection SNR for the PSK

Data Type

int

Minimum Value

0.0

Maximum Value

1.0

Permissions

Read Only

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "PSK_PDSNR int 0" }
```

```
{"Command": "SetValue", "Arguments": "PSK_PDSNR int 1.0 0" }
```

Return:

```
{"PSK_PDSNR": value}
```

PSK_PLL

Description:

Returns the PLL Error

Data Type

float

Minimum Value

0.0

Maximum Value

0.0

Permissions

Read Only

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "PSK_PLL float 0 }
```

```
{"Command": "SetValue", "Arguments": "PSK_PLL float 0.0 0 }
```

Return:

```
{"PSK_PLL": value}
```

PSK_Taps

Description:

Returns the Current Equalizer taps as an array with forward taps concatenated with backwards taps

Data Type

float

Minimum Value

0.0

Maximum Value

0.0

Permissions

Read Only

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "PSK_Taps float 0" }
```

```
{"Command": "SetValue", "Arguments": "PSK_Taps float 0.0 0" }
```

Return:

```
{"PSK_Taps": value}
```

PayloadMode

Description:

BitRate of Payload transmission 0-FH, 1-QPSK5120bps, 2-QPSK2560bps, 3-QPSK1920bps,4-QPSK1280bps, 5-BPSK2560bps, 6-BPSK1280bps, 7-BPSK960bps, 8-BPSK640bps

Data Type

int

Minimum Value

0.0

Maximum Value

8.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "PayloadMode int 0 "}
{"Command": "SetValue", "Arguments": "PayloadMode int 8.0 0 "}
```

Return:

```
{"PayloadMode": value}
```

PeakEnvelopePower

Description:

Peak envelope power of previous transmission

Data Type

float

Minimum Value

0.0

Maximum Value

0.0

Permissions

Read Only

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "PeakEnvelopePower float 0 }
```

```
{"Command": "SetValue", "Arguments": "PeakEnvelopePower float 0.0 0 }
```

Return:

```
{"PeakEnvelopePower": value}
```

PlayMode

Description:

0-Play in Passband; 1 Play in baseband

Data Type

int

Minimum Value

0.0

Maximum Value

1.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "PlayMode int 0 }
```

```
{"Command": "SetValue", "Arguments": "PlayMode int 1.0 0 }
```

Return:

```
{"PlayMode": value}
```

RNG_SpeedOfSound

Description:

Speed of sound in meters per second. Adjust this value for different water salinity etc.

Data Type

float

Minimum Value

340.0

Maximum Value

1600.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "RNG_SpeedOfSound float 0" }  
{"Command": "SetValue", "Arguments": "RNG_SpeedOfSound float 1600.0 0" }  
}
```

Return:

```
{"RNG_SpeedOfSound": value}
```

RNG_TA_DelayMs

Description:

Sets the hold time for ranging in milliseconds. This is the amount of time a modem waits before responding to a range request.

Data Type

int

Minimum Value

2500.0

Maximum Value

8000.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "RNG_TA_DelayMs int 0" }
```

```
{"Command": "SetValue", "Arguments": "RNG_TA_DelayMs int 8000.0 0" }
```

Return:

```
{"RNG_TA_DelayMs": value}
```

RangeTimeout_mS

Description:

Range reply timeout in ms

Data Type

int

Minimum Value

0.0

Maximum Value

60000.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "RangeTimeout_mS int 0 }
```

```
{"Command": "SetValue", "Arguments": "RangeTimeout_mS int 60000.0 0 }
```

Return:

```
{"RangeTimeout_mS": value}
```

RecordMode

Description:

0-Record in Passband; 1 Record in baseband 2 Record Dual Channel

Data Type

int

Minimum Value

0.0

Maximum Value

2.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "RecordMode int 0 "}
{"Command": "SetValue", "Arguments": "RecordMode int 2.0 0 "}
```

Return:

```
{"RecordMode": value}
```

RemoteID

Description:

Local Modem ID 0-254;255 broadcast

Data Type

int

Minimum Value

0.0

Maximum Value

255.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "RemoteID int 0" }
```

```
{"Command": "SetValue", "Arguments": "RemoteID int 255.0 0" }
```

Return:

```
{"RemoteID": value}
```

ReportErroredBytes

Description:

Switch to control whether to report the bytes in a bad CRC packet

Data Type

int

Minimum Value

0.0

Maximum Value

1.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "ReportErroredBytes int 0 }
```

```
{"Command": "SetValue", "Arguments": "ReportErroredBytes int 1.0 0 }
```

Return:

```
{"ReportErroredBytes": value}
```

RxEnable

Description:

enable (1) or disable (0) receiver processing

Data Type

int

Minimum Value

0.0

Maximum Value

1.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "RxEnable int 0" }
```

```
{"Command": "SetValue", "Arguments": "RxEnable int 1.0 0" }
```

Return:

```
{"RxEnable": value}
```

RxScramblerMode

Description:

Scrambler Enable on Rx 0-disable 1-enable

Data Type

int

Minimum Value

0.0

Maximum Value

1.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "RxScramblerMode int 0 }
```

```
{"Command": "SetValue", "Arguments": "RxScramblerMode int 1.0 0 }
```

Return:

```
{"RxScramblerMode": value}
```

SNR

Description:

SNR Estimate

Data Type

float

Minimum Value

0.0

Maximum Value

0.0

Permissions

Read Only

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "SNR float 0" }
```

```
{"Command": "SetValue", "Arguments": "SNR float 0.0 0" }
```

Return:

```
{"SNR": value}
```

SSB_Enable

Description:

SSB Enable, default 0

Data Type

int

Minimum Value

0.0

Maximum Value

1.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "SSB_Enable int 0 }
```

```
{"Command": "SetValue", "Arguments": "SSB_Enable int 1.0 0 }
```

Return:

```
{"SSB_Enable": value}
```

SSB_NREnable

Description:

SSB Enable advanced squelch, AGC and Noise Reduction

Data Type

int

Minimum Value

0.0

Maximum Value

1.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "SSB_NREnable int 0" }
```

```
{"Command": "SetValue", "Arguments": "SSB_NREnable int 1.0 0" }
```

Return:

```
{"SSB_NREnable": value}
```

SSB_SetPTT

Description:

SSB set state of ptt from console

Data Type

int

Minimum Value

0.0

Maximum Value

1.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "SSB_SetPTT int 0 }
```

```
{"Command": "SetValue", "Arguments": "SSB_SetPTT int 1.0 0 }
```

Return:

```
{"SSB_SetPTT": value}
```

SSB_SqLevel

Description:

SSB Squelching threshold 0-always on, default=.005

Data Type

float

Minimum Value

0.0

Maximum Value

1.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "SSB_SqLevel float 0 }
```

```
{"Command": "SetValue", "Arguments": "SSB_SqLevel float 1.0 0 }
```

Return:

```
{"SSB_SqLevel": value}
```

SSB_Txpower

Description:

SSB Output power scale, default=1

Data Type

float

Minimum Value

0.0

Maximum Value

100.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "SSB_Txpower float 0 }
```

```
{"Command": "SetValue", "Arguments": "SSB_Txpower float 100.0 0 }
```

Return:

```
{"SSB_Txpower": value}
```

SSB_UseGPIO

Description:

Process J7 P3 GPIO 8/6 and P9 GPIO 7/14 Switches, default=0

Data Type

int

Minimum Value

0.0

Maximum Value

1.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "SSB_UseGPIO int 0 }
```

```
{"Command": "SetValue", "Arguments": "SSB_UseGPIO int 1.0 0 }
```

Return:

```
{"SSB_UseGPIO": value}
```

SSB_Volume

Description:

SSB Speaker Volume, default=1

Data Type

float

Minimum Value

0.0

Maximum Value

100.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "SSB_Volume float 0 }
```

```
{"Command": "SetValue", "Arguments": "SSB_Volume float 100.0 0 }
```

Return:

```
{"SSB_Volume": value}
```

SSB_VxHang

Description:

SSB Vox hangover switch time seconds, default=2.0

Data Type

float

Minimum Value

0.0

Maximum Value

10.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "SSB_VxHang float 0 }
```

```
{"Command": "SetValue", "Arguments": "SSB_VxHang float 10.0 0 }
```

Return:

```
{"SSB_VxHang": value}
```

SSB_VxLevel

Description:

SSB Vox switching threshold 0-always on, default=.005

Data Type

float

Minimum Value

0.0

Maximum Value

1.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "SSB_VxLevel float 0 }
```

```
{"Command": "SetValue", "Arguments": "SSB_VxLevel float 1.0 0 }
```

Return:

```
{"SSB_VxLevel": value}
```

SSB_VxMode

Description:

SSB 1-Enable voice activated PTT (vox), 0-disable

Data Type

int

Minimum Value

0.0

Maximum Value

1.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "SSB_VxMode int 0 }
```

```
{"Command": "SetValue", "Arguments": "SSB_VxMode int 1.0 0 }
```

Return:

```
{"SSB_VxMode": value}
```

SSB_carrier

Description:

SSB set Tx Rx carrier

Data Type

int

Minimum Value

22000.0

Maximum Value

35000.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "SSB_carrier int 0 }
```

```
{"Command": "SetValue", "Arguments": "SSB_carrier int 35000.0 0 }
```

Return:

```
{"SSB_carrier": value}
```

SSB_sideband

Description:

SSB sidband 0-L 1-U

Data Type

int

Minimum Value

0.0

Maximum Value

1.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "SSB_sideband int 0 }
```

```
{"Command": "SetValue", "Arguments": "SSB_sideband int 1.0 0 }
```

Return:

```
{"SSB_sideband": value}
```

SignalEnergy

Description:

Signal Energy Measured during last FH acquisition

Data Type

float

Minimum Value

0.0

Maximum Value

1.0

Permissions

Read Only

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "SignalEnergy float 0 }
```

```
{"Command": "SetValue", "Arguments": "SignalEnergy float 1.0 0 }
```

Return:

```
{"SignalEnergy": value}
```

StreamingTxLen

Description:

Size of superpacket when streaming or uploading.

Data Type

int

Minimum Value

0.0

Maximum Value

8192.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "StreamingTxLen int 0 }
```

```
{"Command": "SetValue", "Arguments": "StreamingTxLen int 8192.0 0 }
```

Return:

```
{"StreamingTxLen": value}
```

SystemMode

Description:

System Mode 0-DataModem, 1-SSB , 2-DataModem+SSB, 3-SMA IN to Tx

Data Type

int

Minimum Value

0.0

Maximum Value

4.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "SystemMode int 0 "}
{"Command": "SetValue", "Arguments": "SystemMode int 4.0 0 "}
```

Return:

```
{"SystemMode": value}
```

TCPEcho

Description:

0-disable TCP echo in telnet Tx stream; 1 enable TCP echo in telnet Tx stream

Data Type

int

Minimum Value

0.0

Maximum Value

1.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "TCPEcho int 0 "}
{"Command": "SetValue", "Arguments": "TCPEcho int 1.0 0 "}
```

Return:

```
{"TCPEcho": value}
```

Temp_Ambient

Description:

Ambient bottle temperature in degrees C

Data Type

float

Minimum Value

0.0

Maximum Value

0.0

Permissions

Read Only

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "Temp_Ambient float 0 }
```

```
{"Command": "SetValue", "Arguments": "Temp_Ambient float 0.0 0 }
```

Return:

```
{"Temp_Ambient": value}
```

TxChirpMode

Description:

Transmit chirps prior to packets 0-disable 1-enable

Data Type

int

Minimum Value

0.0

Maximum Value

1.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "TxChirpMode int 0 }
```

```
{"Command": "SetValue", "Arguments": "TxChirpMode int 1.0 0 }
```

Return:

```
{"TxChirpMode": value}
```

TxEnable

Description:

enable (1) or disable (0) transmit processing

Data Type

int

Minimum Value

0.0

Maximum Value

1.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "TxEnable int 0 }
```

```
{"Command": "SetValue", "Arguments": "TxEnable int 1.0 0 }
```

Return:

```
{"TxEnable": value}
```

TxPower

Description:

Tx power in watts

Data Type

int

Minimum Value

0.0

Maximum Value

100.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "TxPower int 0" }
```

```
{"Command": "SetValue", "Arguments": "TxPower int 100.0 0" }
```

Return:

```
{"TxPower": value}
```

TxPowerWatts

Description:

TX output power in watts

Data Type

float

Minimum Value

0.0

Maximum Value

1000.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "TxPowerWatts float 0 }
```

```
{"Command": "SetValue", "Arguments": "TxPowerWatts float 1000.0 0 }
```

Return:

```
{"TxPowerWatts": value}
```

TxTimeout_mS

Description:

Transmit timeout in ms

Data Type

int

Minimum Value

0.0

Maximum Value

60000.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "TxTimeout_mS int 0" }
```

```
{"Command": "SetValue", "Arguments": "TxTimeout_mS int 60000.0 0" }
```

Return:

```
{"TxTimeout_mS": value}
```

UPCONVERT_Carrier

Description:

Upconverter Carrier Frequency in Hz

Data Type

int

Minimum Value

20000.0

Maximum Value

59750.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "UPCONVERT_Carrier int 0 "}
{"Command": "SetValue", "Arguments": "UPCONVERT_Carrier int 59750.0 0"}
}
```

Return:

```
{"UPCONVERT_Carrier": value}
```

UPCONVERT_OutputScale

Description:

Upconverter Output Scale

Data Type

float

Minimum Value

0.0

Maximum Value

10.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "UPCONVERT_OutputScale float 0 }  
{"Command": "SetValue", "Arguments": "UPCONVERT_OutputScale float 10.0  
0 }
```

Return:

```
{"UPCONVERT_OutputScale": value}
```

UnitTestMode

Description:

0 or 1: If 1 Pump data in a deterministic way from a socket. This disables timeouts on recordings and Play

Data Type

int

Minimum Value

0.0

Maximum Value

1.0

Permissions

Read and Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "UnitTestMode int 0 }
```

```
{"Command": "SetValue", "Arguments": "UnitTestMode int 1.0 0 }
```

Return:

```
{"UnitTestMode": value}
```

brdState

Description:

Board State State

Data Type

int

Minimum Value

0.0

Maximum Value

0.0

Permissions

Read Only

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "brdState int 0 }
```

```
{"Command": "SetValue", "Arguments": "brdState int 0.0 0 }
```

Return:

```
{"brdState": value}
```

rxState

Description:

Present Receiver State

Data Type

int

Minimum Value

0.0

Maximum Value

0.0

Permissions

Read Only

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "rxState int 0 }
```

```
{"Command": "SetValue", "Arguments": "rxState int 0.0 0 }
```

Return:

```
{"rxState": value}
```

tpaState

Description:

Power Amplifier State

Data Type

int

Minimum Value

0.0

Maximum Value

0.0

Permissions

Read Only

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "tpaState int 0 }
```

```
{"Command": "SetValue", "Arguments": "tpaState int 0.0 0 }
```

Return:

```
{"tpaState": value}
```

24 TIPE

24.1 TIPE Commands

pshell Command: range

Description:

Sends a two-way range request using approximately <Power> watts. This command issues a range request and sends it to the specified module (modem, echosounder, transponder) at the configured remoteID. The remote module holds the request for a predetermined amount of time, and then replies with a range response. Popoto will then send back a range report consisting of the distance between the devices, the configured speed of sound, and the computed round trip time.

Modem: The first modem requests a range response, and the second modem sends a range reply after it receives the request. The round trip time is computed, and then scaled with the configured speed of sound to come up with a time of flight range.

Transponder: This algorithm works similarly to the modem; however, rather than exchanging digital messages, the signaling is done via tones.

Echosounder: The echosounder sends a chirp (as configured in the TIPE configuration settings) and listens for its echo reflection to determine the distance to the surface or bottom.

Note that the speed of sound, and the ranging hold time are configurable parameters. If you change the ranging hold time, it is imperative that you configure both the local and remote modules to have the same hold time. Otherwise, Popoto will give erroneous range reports.

Invocation

```
range <power> [module]
```

Examples

```
range 20  
range 20 transponder  
range 20 echosounder  
{ "Range":500.002441,"Roundtrip Delay":666.669922,"SpeedOfSound":1500.000000,"Units":"m,  
ms, meters per second"}
```

```
Or if there is a timeout  
{ "Alert":"Timeout"}
```

pshell Command: ping

Description:

Send an acoustic test message. This API sends the text "Popoto Test Message" using the configured data rate and the approximate specified power level when the module is set to 'modem'. For the 'pinger' module, it sends a tonal ping as configured with the 'pinger:<configuration variables>'. It is important to note that calling ping with a power level latches that power level in the transmitter, to be used for subsequent transmissions.

Modem (default): Sends the text "Popoto Test Message."

Pinger: Sends a tonal ping as configured with the 'pinger:<configuration variables>'.

Invocation

```
ping <power level> [module]
```

Examples

```
ping 10
```

```
ping 10 pinger
```

Sends a test message (Popoto Test Message) using approximately 10 watts of power

24.2 TIPE Variables

echosounder:mode

Description:

Echo Sounder Modes of operation. Default value: 0 = Off).

Data Type

int

Minimum Value

0

Maximum Value

1

Permissions

Read Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "echosounder:mode int 0 }  
{"Command": "SetValue", "Arguments": "echosounder:mode int 1 0 }
```

Return:

```
{"echosounder:mode": value}
```

echosounder:carrier

Description:

Center frequency of echo sounder. Default value: 10000.

Data Type

int

Minimum Value

5000

Maximum Value

45000

Permissions

Read Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "echosounder:carrier int 0 "}
{"Command": "SetValue", "Arguments": "echosounder:carrier int 45000 0"}
}
```

Return:

```
{"echosounder:carrier": value}
```

echosounder:hfmchirpbw

Description:

HFM chirp bandwidth. Default value: 5 khz.

Data Type

int

Minimum Value

1000

Maximum Value

8000

Permissions

Read Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "echosounder:hfmchirpbw int 0 }  
{"Command": "SetValue", "Arguments": "echosounder:hfmchirpbw int 8000  
0 }
```

Return:

```
{"echosounder:hfmchirpbw": value}
```

echosounder:hfmchirplength

Description:

HFM chirp length. Default value: 0.032.

Data Type

float

Minimum Value

0.1

Maximum Value

0.01

Permissions

Read Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "echosounder:hfmchirplength float  
0 } }
```

```
{"Command": "SetValue", "Arguments": "echosounder:hfmchirplength float  
0.01 0 } }
```

Return:

```
{"echosounder:hfmchirplength": value}
```

echosounder:timeout

Description:

Echo sounder timeout. Default value: 10.

Data Type

float

Minimum Value

1

Maximum Value

60

Permissions

Read Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "echosounder:timeout float 0 }
```

```
{"Command": "SetValue", "Arguments": "echosounder:timeout float 60 0 }
```

Return:

```
{"echosounder:timeout": value}
```

echosounder:speedofsound

Description:

Speed of sound. Default value: 1500.

Data Type

float

Minimum Value

300

Maximum Value

2000

Permissions

Read Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "echosounder:speedofsound float 0  
}
```

```
{"Command": "SetValue", "Arguments": "echosounder:speedofsound float  
2000 0 }
```

Return:

```
{"echosounder:speedofsound": value}
```

transponder:mode

Description:

Transponder modes of operation. Default value: 0 = Off.

Data Type

int

Minimum Value

0

Maximum Value

1

Permissions

Read Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "transponder:mode int 0 }
```

```
{"Command": "SetValue", "Arguments": "transponder:mode int 1 0 }
```

Return:

```
{"transponder:mode": value}
```

transponder:carrier

Description:

Transponder carrier frequency. Default value: 8087.

Data Type

float

Minimum Value

45000

Maximum Value

5000

Permissions

Read Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "transponder:carrier float 0 "}
{"Command": "SetValue", "Arguments": "transponder:carrier float 5000 0 "}
}
```

Return:

```
{"transponder:carrier": value}
```

transponder:detecttonef

Description:

Detect tone frequency. Default value: 1997.

Data Type

float

Minimum Value

-5000

Maximum Value

5000

Permissions

Read Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "transponder:detecttonef float 0" }
```

```
{"Command": "SetValue", "Arguments": "transponder:detecttonef float 5000 0" }
```

Return:

```
{"transponder:detecttonef": value}
```

transponder:detecttonelength

Description:

Detect tone length. Default value: 0.032.

Data Type

float

Minimum Value

0.01

Maximum Value

0.1

Permissions

Read Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "transponder:detecttonelength  
float 0 }
```

```
{"Command": "SetValue", "Arguments": "transponder:detecttonelength  
float 0.1 0 }
```

Return:

```
{"transponder:detecttonelength": value}
```

transponder:turnaroundtime

Description:

Turnaround time. Default value: 0.116.

Data Type

float

Minimum Value

0.01

Maximum Value

5

Permissions

Read Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "transponder:turnaroundtime float  
0 }
```

```
{"Command": "SetValue", "Arguments": "transponder:turnaroundtime float  
5 0 }
```

Return:

```
{"transponder:turnaroundtime": value}
```

transponder:respondtonef

Description:

Respond tone frequency. Default value: 1251.

Data Type

float

Minimum Value

-5000

Maximum Value

5000

Permissions

Read Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "transponder:respondtonef float 0  
}
```

```
{"Command": "SetValue", "Arguments": "transponder:respondtonef float  
5000 0 }
```

Return:

```
{"transponder:respondtonef": value}
```

transponder:respondtoneLength

Description:

Respond tone length. Default value: 0.032.

Data Type

float

Minimum Value

0.01

Maximum Value

0.1

Permissions

Read Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "transponder:respondtoneLength  
float 0 }
```

```
{"Command": "SetValue", "Arguments": "transponder:respondtoneLength  
float 0.1 0 }
```

Return:

```
{"transponder:respondtoneLength": value}
```

transponder:timeout

Description:

Transponder timeout. Default value: 10.

Data Type

float

Minimum Value

1

Maximum Value

60

Permissions

Read Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "transponder:timeout float 0 }
```

```
{"Command": "SetValue", "Arguments": "transponder:timeout float 60 0 }
```

Return:

```
{"transponder:timeout": value}
```

pinger:mode

Description:

Pinger mode. Default value: 0 = Off.

Data Type

int

Minimum Value

0

Maximum Value

1

Permissions

Read Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "pinger:mode int 0" }
```

```
{"Command": "SetValue", "Arguments": "pinger:mode int 1 0" }
```

Return:

```
{"pinger:mode": value}
```

pinger:carrier

Description:

Pinger carrier frequency. Default value: 8087.

Data Type

float

Minimum Value

5000

Maximum Value

45000

Permissions

Read Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "pinger:carrier float 0 }
```

```
{"Command": "SetValue", "Arguments": "pinger:carrier float 45000 0 }
```

Return:

```
{"pinger:carrier": value}
```

pinger:pingtone

Description:

Ping tone frequency. Default value: -1000.

Data Type

float

Minimum Value

-5000

Maximum Value

5000

Permissions

Read Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "pinger:pingtone float 0 }
```

```
{"Command": "SetValue", "Arguments": "pinger:pingtone float 5000 0 }
```

Return:

```
{"pinger:pingtone": value}
```

pinger:tonelength

Description:

Tone length. Default value: 0.014.

Data Type

float

Minimum Value

0.01

Maximum Value

0.1

Permissions

Read Write

JSON API Syntax:

```
{"Command": "GetValue", "Arguments": "pinger:tonelength float 0 }
```

```
{"Command": "SetValue", "Arguments": "pinger:tonelength float 0.1 0 }
```

Return:

```
{"pinger:tonelength": value}
```


25 Appendix A

25.1 The Acoustic Message Header

Every acoustic packet contains a header packet. Some types of acoustic packets are only a header, while others contain a subsequent payload packet. bits Field

Table 25.1: Header packet format

Bits	Field	Purpose	Format
0-7	Message Type	Identify between Packet, Packet with payload, ranging etc. MessageIDs 0 - Data 128 -Range Response 129 -Range Request 130 -Status	8 Bit MessageID
8-15	SenderID	ID of the transmitting modem	0x0 – 0xfe - ID 0xff Broadcast
16-23	ReceiverID	The intended ID of the destination receiver.	0x0 – 0xfe - ID 0xff = Broadcast message
24-31	TxPower	The transmitted scale factor as entered by the transmitting modem	Transmit power level as a Q8 scale value
32-47	PayloadInfo	If the present message does not contain a payload, then this field is 0. If a payload follows, the bits are assembled according to the payloadinfo fields described below.	See Section 25.2

25.2 Payload Structure

If header bytes 4 and 5 are not zero, modulated payload data will immediately follow the modulated header data. The payload is described by the 16 bits in the payload info field of the header as follows:

Table 25.2: Header Byte 4

7	6	5	4	3	2	1	0
Plen7	Plen6	Plen5	Plen4	Plen3	Plen2	Plen1	Plen0

Table 25.3: Header Byte 5

15	14	13	12	11	10	9	8
Mod4	Mod3	Mod2	Mod1	Mod0	Stream	Plen9	Plen8

The length of the payload in bytes is set by the 10 bits of the Plen field. Although the field contains 10 bits, the payload size is capped by the software to a maximum of 256 bytes. Bits 11-15 of byte 5 of the header contain the modulation employed for the payload. Popoto uses the following enumerated modulations:

Table 25.4: Modulation Types (Mod Values)

Modulation Bitfield	Modulation Scheme	Data Rate
0	Frequency Hopped FSK	80 bps
1	Phase Shift Keying	5120 bps
2	Phase Shift Keying	2560 bps
3	Phase Shift Keying	1280 bps
4	Phase Shift Keying	640 bps
5	Phase Shift Keying	10240(uncoded) bps

When large files are transmitted, it is more efficient to transfer the file in streaming mode. When streaming mode is invoked, the bit 10, of header byte 5 is set to indicate streaming mode. In streaming mode, the payload length Plen indicates the number of 255 byte frames which follow before another header transmission. All 255 byte packet remainders are handled by the software automatically.

26 Appendix B

26.1 Assembly Drawings

4

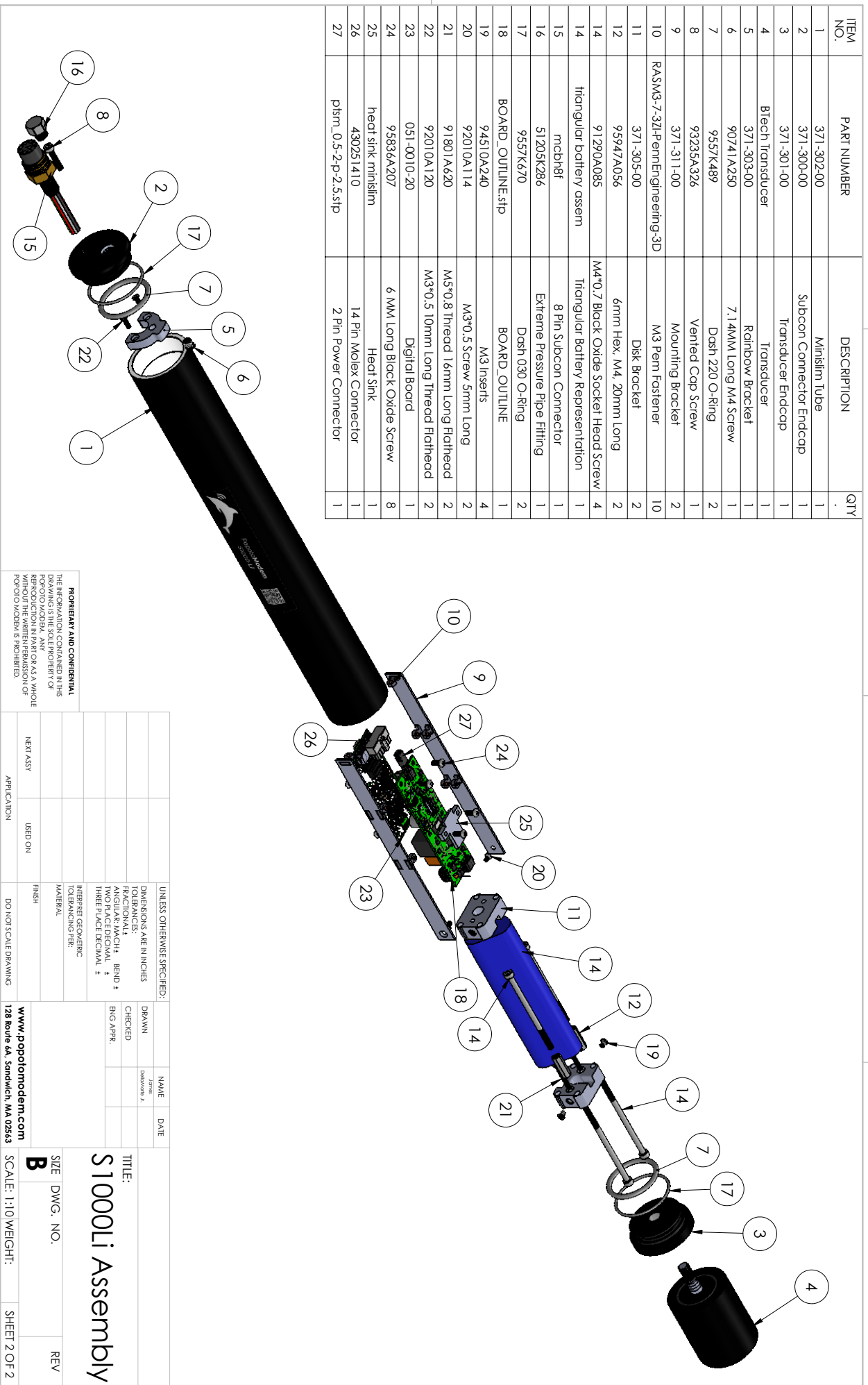
3

2

1

ITEM NO.	PART NUMBER	DESCRIPTION	QTY
1	371-302-00	Minislilm Tube	1
2	371-300-00	Subcon Connector Endcap	1
3	371-301-00	Transducer Endcap	1
4	Btech Transducer	Transducer	1
5	371-303-00	Rainbow Bracket	1
6	90741A250	7.14MM Long M4 Screw	1
7	9557K489	Dash 220 O-Ring	2
8	93235A326	Vented Cap Screw	1
9	371-311-00	Mounting Bracket	2
10	RASM3-7-32L-PennEngineering-3D	M3 Penn Fastener	10
11	371-305-00	Disk Bracket	2
12	95947A056	6mm Hex, M4, 20mm Long	2
14	91290A085	M4*0.7 Black Oxide Socket Head Screw	4
14	triangular battery assem	Triangular Battery Representation	1
15	mcb18f	8 Pin Subcon Connector	1
16	51205K286	Extreme Pressure Pipe Fitting	1
17	9557K670	Dash 030 O-Ring	2
18	BOARD_OUTLINE.stp	BOARD_OUTLINE	1
19	94510A240	M3 Inserts	4
20	92010A114	M3*0.5 Screw 5mm Long	2
21	91801A620	M5*0.8 Thread 16mm Long Flathead	2
22	92010A120	M3*0.5 10mm Long Thread Flathead	2
23	051-0010-20	Digital Board	1
24	95836A207	6 MM Long Black Oxide Screw	8
25	heat sink minislilm	Heat Sink	1
26	430251410	14 Pin Molex Connector	1
27	ps1m_0.5-2-p-2.5.stp	2 Pin Power Connector	1

B



A

PROPRIETARY AND CONFIDENTIAL
 THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF POPOTO MODERN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF POPOTO MODERN IS PROHIBITED.

UNLESS OTHERWISE SPECIFIED:	NAME	DATE
DIMENSIONS ARE IN INCHES	DATE	
TOLERANCES:	DRAWN	
FRACTIONAL: ±	CHECKED	
ANGULAR: MACH ± BEND ±	ENG APPR.	
TWO PLACE DECIMAL ±		
THREE PLACE DECIMAL ±		
INTERPRET GEOMETRIC TOLERANCING PER: MATERIAL		

TITLE: **S1000Li Assembly**
 SIZE: **B** DWG. NO. **B** REV
 SCALE: 1:10 WEIGHT: SHEET 2 OF 2
 www.popotomodern.com
 128 Route 6A, Sandwich, MA 02563

4

3

2

1