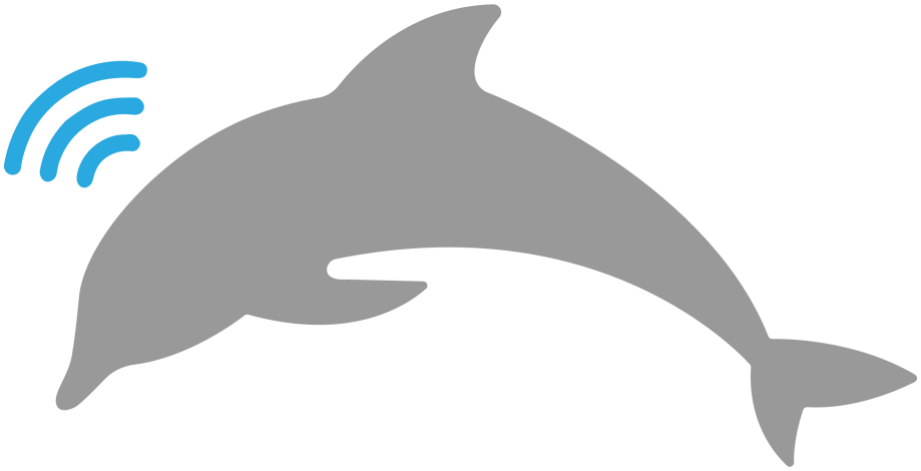
Popoto Acoustic Modem
Python API Manual



Document Versioning Information			
Version	**Purpose**	**Author**	**Date**
.1	Initial draft	John DellaMorte	11/28/19
.2	Added Python and Matlab	John DellaMorte	12/3/2019
.3	Update formatting for Python		1/10/2020

Introduction

All status and control of Popoto modems happens over sockets using JSON messages. These API messages facilitate a human readable format for the user to design custom control software for Popoto. This control is accomplished with high level languages such as Matlab, Python, or C++. Because socket methods are ubiquitous in these high level languages, programming for this modem is a simple matter of sending the desired JSON messages to Popoto using the desired high level language.

Highlevel Description of Popoto API

Sockets

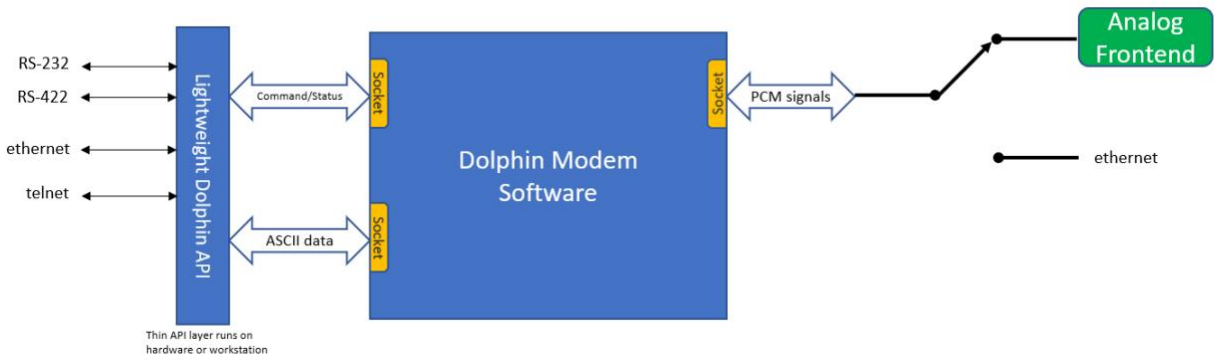
The IO to the Popoto software supports IP Sockets. Even the analog signal data supports the socket IO. This provides great flexibility for interface, test, software portability, and software test.

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 specified by the IP address of the Popoto modem as set in the TBD file in the root of the SD card of the hardware. 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 Port (Not for typical use)

For the purpose of this API document, the Command port is the port used. Connection to the hardware is done by standard Python socket connection methods.



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.

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:

```
{"Command": "<command>", "Arguments": "<argument>"}
```

Example: *Get the software version*

An example of the 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 2.3.3 99"}
```

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.

System Level Variables

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

Datatypes

The system level variables are fall into two types:

int	32 bit signed integer
float	32 bit IEEE floating point value

Anytime the system variables are accessed the datatype must be specified in the argument section of the command.

Example: *Set the carrier frequency*

An example of the a simple JSON to set the carrier frequency of the modem is shown in this example. Note that to change the modem we need to change the carrier of both the upconvertor and downconvertor baseband conversion blocks. This means setting two system level variables. In this example the carrier is set to 25000 KHz.

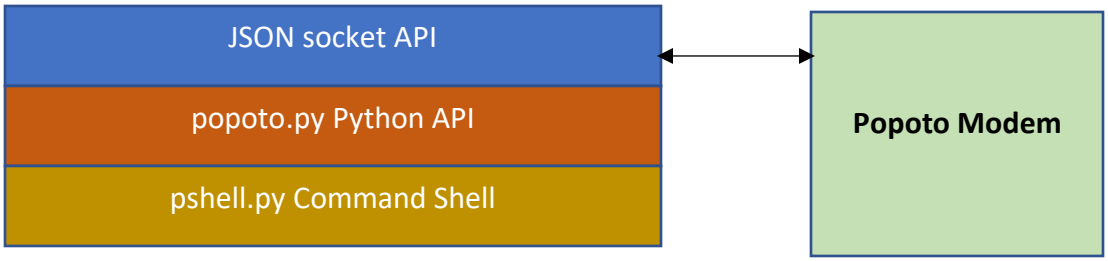
```
{ "Command": "SetValue", "Arguments": "UPCONVERT_Carrier int 25000 0"}
{ "Command": "SetValue", "Arguments": "DOWNCONVERT_Carrier int 25000 0"}
```

And would result in the modem responding as follows:

```
{"Info":"Value Set UPCONVERT_Carrier=25000"}
{"Info":"Value Set DOWNCONVERT_Carrier=25000"}
```

The Three Levels of API

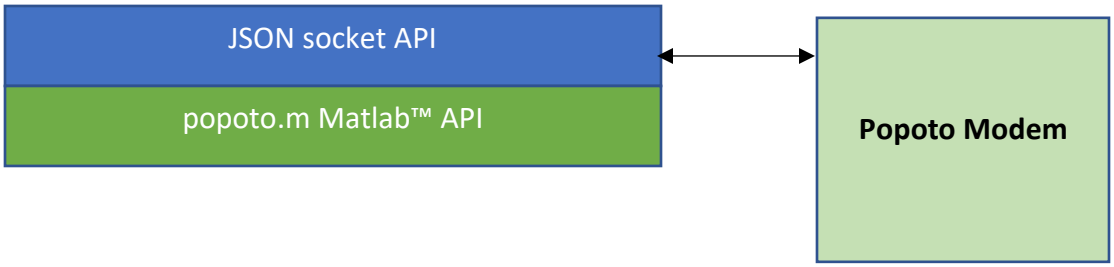
The primary interface to Popoto is through TCP socket IO using JSON messages. In addition to this API there exists popoto.m which is a python library that facilitates building the JSON messages to control the modem. Additionally there is a python program called pshell.py that contains useful shell commands for controlling the modem.



This library and pshell utilize all the same system variables as described in the Popoto JSON Socket API.

It is useful to note that pshell.py is filled with examples of interfacing to the Python Popoto API. The user is encouraged to add new customized commands to pshell.py to simplify and automate their particular use case.

Additionally Popoto Modem includes a standalone Matlab™ API called popoto.m.



Popoto Socket API Reference

Commands

A Command is a type of JSON message sent to Popoto to control the modem. The various Popoto Commands are described below.

{"Command": "DisableMSMLog", "Arguments": "Unused Arguments"} **This command disables logging of the modem state machine information into the modem logs.**	**SYS**
{"Command": "EnableMSMLog", "Arguments": "Unused Arguments"} **This command enables logging of the modem state machine information into the modem logs.**	**SYS**
{"Command": "Event_powerDown", "Arguments": "arg"} **This command places the modem in powerdown mode 1 or 2** **1- Deep Sleep awakened by timer or interrupt** **2- Powerdown awakened by MSP**	**SYS**
{"Command": "Event_StartRx", "Arguments": "Unused Arguments"} **The Event_StartRx command places the modem in receive mode.**	**SYS**
{"Command": "GetValue", "Arguments": "arg"} **The GetValue command retrieves a parameter given by arg. The argument typically contains the <parameter name> <parameter type> <channel>**	**SYS**
{"Command": "GetVersion", "Arguments": "Unused Arguments"} **The GetVersion command queries the current software version.**	**SYS**
{"Command": "SetValue", "Arguments": "arg"} **The SetValue command sets a parameter given by arg. The argument typically contains the <parameter name> <parameter type> <value> <channel>**	**SYS**
{"Command": "startVoice", "Arguments": "Unused Arguments"} **The startVoice command sets the system into voice SSB mode.**	**SYS**

{"Command": "StartRecording", "Arguments": "<local filename> <duration in seconds>"} **This command initiates a recording of baseband or passband data to SD card**	**Rx**
{"Command": "StopRecording", "Arguments": "Unused Arguments"} **This command terminates a recording of baseband or passband data**	**Rx**

{"Command": "Event_playPcmQueueEmpty", "Arguments": Unused Arguments"} **This command finishes the StartNetPlay command.**	**Tx**
{"Command": "Event_sendTestPacket", "Arguments":"Unused Arguments" } **This command initiates a playing of baseband or passband data**	**Tx**
{"Command": "Event_startTxCal", "Arguments": "Unused Arguments "} **This command initiates a transducer self calibration. Set output power to 1 watt first using TxPowerWatts variable.**	**Tx**
{"Command": "StartNetPlay", "Arguments": "0 0 "} **This command initiates a playing of baseband or passband data out the transducer directly from the socket.**	**Tx**
{"Command": "StartPlaying", "Arguments": "<local filename> <duration in seconds>"} **This command initiates a playback of baseband or passband data from SD card.**	**Tx**
{"Command": "StopPlaying", "Arguments": "Unused Arguments"} **This command terminates a playback of baseband or passband data from SD card.**	**Tx**
{"Command": "TransmitData", "Arguments": [b0,b1,b2..bn]} **This command sends byte data to a remote modem.**	**Tx**
{"Command": "Event_sendRanging", "Arguments": "Unused Arguments"} **This command initiates a range request to a remote modem.**	**Tx/Rx**

Keyword Responses

In addition to Alerts, popoto modems sends various JSON messages back to the user delimited by JSON keywords. These messages are listed below:

INFO

The INFO keyword identifies requested information being sent back by Popoto immediately following a query to Popoto.

Example:
{"Info ":"Popoto Modem Version 2.3.3 99"}

ALERT

An alert is a JSON message sent by Popoto when potentially interesting asynchronous events happen on the modem. Alerts may be parsed as JSON messages by a user to advance through states in a user application. . The various Popoto Alerts are described below.

{“Alert","CRCError"}

This alert is sent when a CRC failure occurred when processing either the header or the payload.

{"Alert","CRCCheck"}

This alert is sent when a CRC check has passed when processing either the header or the payload.

{"Alert","TxComplete"}

This alert is sent when a transmit packet has finished sending.

{"Alert","RangeError"}

This alert is sent when a ranging cycle could not be completed. This usually happens when no response is received or a range response was received in error.

{"Alert","Timeout"}

This alert is sent when a timeout happens. This timeout can either be a transmit timeout or if the range window expires without receiving a range response.

Header

This keyword is used to identify the receipt of valid header data.

Example:

```
{"Header":[0,10,255,25,32,0,0,210,0]}
```

Data

This keyword is used to identify the receipt of valid payload data.

Example:

```
{"Data":[68,111,108,112,104,105,110,32,84,101,115,116,32,77,101,115,115,97,1
03,101,32,72,105,103,104,32,83,112,101,101,100,33]}
```

Doppler

This keyword is used to identify the doppler estimate when a header is received.

Example:

```
{"DopplerVelocity":0.000000}
```

SNR

This keyword is used to identify the SNR estimate when a header is received.

Example:

```
{"SNR":8.000000}
```

PDSNR

This keyword is used to identify the post detection PSK SNR estimate in a received payload.

Example:

```
{"PDSNR":8.000000}
```

System Level Variables

Variable Name:	APP_ModemSMAOut
Description: **This value sets whether to send signal data out the SMA port.**	
Genre:	System
Data Type:	int
Permissions:	Read/Write
Min :	0- Not out SMA
Max :	1- Sent out SMA
Syntax **:** **{ "Command": "GetValue", "Arguments": "APP_ModemSMAOut int 0"}** **{ "Command": "SetValue", "Arguments": "APP_ModemSMAOut int 1 0"}**	
Return : **{"APP_ModemSMAOut":value}**	

Variable Name:	APP_SocketBasedPCM
Description: **This value sets whether to enable socket based PCM or default of A/D D/A PCM**	
Genre:	System
Data Type:	int
Permissions:	Read/Write
Min :	0- A/D D/A based PCM
Max :	1- Socket based PCM
Syntax **:** **{ "Command": "GetValue", "Arguments": "APP_ SocketBasedPCM int 0"}** **{ "Command": "SetValue", "Arguments": "APP_ SocketBasedPCM int 1 0"}**	
Return : **{"APP_ SocketBasedPCM":value}**	

System Level Variables (cont)

Variable Name:	APP_SystemMode
Description: **This value sets whether the modem is in 0-data mode, 1-SSB Tx, 2- SSB Rx**	
Genre:	System
Data Type:	int
Permissions:	Read/Write
Min :	0- data mode
Max :	1- SSB Tx mode 2- SSB Rx mode
Syntax **:** **{ "Command": "GetValue", "Arguments": "APP_ SystemMode int 0"}** **{ "Command": "SetValue", "Arguments": "APP_ SystemMode int 1 0"}**	
Return : **{"APP_ SystemMode":value}**	

Variable Name:	BatteryVoltage
Description: **This command queries the system battery voltage.**	
Genre:	System
Data Type:	float
Permissions:	Read
Min :	0.0 Volts
Max :	40.0 Volts
Syntax **:** **{ "Command": "GetValue", "Arguments": "BatteryVoltage float 0"}**	
Return : **{"BatteryVoltage":*voltage*}**	

System Level Variables (cont)

Variable Name:	LedEnable
Description: **This value determines if onboard LEDS are 0-disabled, or 1-enabled. It can be useful for power reduction to summarily disable board LEDS.**	
Genre:	System
Data Type:	int
Permissions:	Read/Write
Min :	0- LEDS disabled
Max :	1- LEDS enabled
Syntax **:** **{ "Command": "GetValue", "Arguments": "LedEnable int 0"}** **{ "Command": "SetValue", "Arguments": "LedEnable int 1 0"}**	
Return : **{"LedEnable":value}**	

Variable Name:	LoggingLevel
Description: **This value sets the level of verbosity for the message logging**	
Genre:	System
Data Type:	int
Permissions:	Read/Write
Min :	0- minimum logging
Max :	5- maximum logging
Syntax **:** **{ "Command": "GetValue", "Arguments": "LoggingLevel int 0"}** **{ "Command": "SetValue", "Arguments": "LoggingLevel int 1 0"}**	
Return : **{"LoggingLevel":value}**	

System Level Variables (cont)

Variable Name:	RNG_SpeedOfSound
Description: **This value sets the speed of sound in meters per second. An accurate measure of the speed of sound is necessary for accurate ranging functionality. Please set this value as a function of the local environment. The default is 1500 meters per second.**	
Genre:	System
Data Type:	float
Permissions:	Read/Write
Min :	1400 mps
Max :	1600 mps
Syntax **:** **{ "Command": "GetValue", "Arguments": " RNG_SpeedOfSound float 0"}** **{ "Command": "SetValue", "Arguments": " RNG_SpeedOfSound float 1500 0"}**	
Return : **{" RNG_SpeedOfSound":value}**	

Variable Name:	TCPecho
Description: **This value determines if TCP characters are echoed on the telnet Tx stream**	
Genre:	System
Data Type:	int
Permissions:	Read/Write
Min :	0- no echo
Max :	1- telnet echo
Syntax **:** **{ "Command": "GetValue", "Arguments": "TCPecho int 0"}** **{ "Command": "SetValue", "Arguments": "TCPecho int 1 0"}**	
Return : **{"TCPecho":value}**	

System Level Variables (cont)

Variable Name:	Temp_Ambient
Description: **This command queries the system local ambient temperature in degrees Celsius.**	
Genre:	System
Data Type:	float
Permissions:	Read
Val :	Degrees C
Syntax **:** **{ "Command": "GetValue", "Arguments": "Temp_Ambient float 0"}**	
Return : **{"Temp_Ambient":*voltage*}**	

Receiver Oriented Variables

Variable Name:	DOWNCONVERT_Carrier
Description: **This value sets the receiver downconverter carrier in Hz.**	
Genre:	Receiver
Data Type:	int
Permissions:	Read/Write
Min :	20000
Max :	59750
Syntax **:** **{ "Command": "GetValue", "Arguments": "DOWNCONVERT_Carrier int 0"}** **{ "Command": "SetValue", "Arguments": "DOWNCONVERT_Carrier int 25000 0"}**	
Return : **{"DOWNCONVERT_Carrier":value}**	

Variable Name:	FHDEMOD_DetectThresholdDB
Description: **This value sets the detection threshold for acquisition. The default is 160 which corresponds to about -5 Db. The lower the value the more sensitive the detection but the larger the false alarm detection will be. It is generally not recommended to modify this from the default.**	
Genre:	Receiver
Data Type:	float
Permissions:	Read/Write
Min :	0.
Max :	300.
Syntax **:** **{ "Command": "GetValue", "Arguments": "FHDEMOD_DetectThresholdDB float 0"}** **{ "Command": "SetValue", "Arguments": "FHDEMOD_DetectThresholdDB float 160 0"}**	
Return : **{"FHDEMOD_DetectThresholdDB":value}**	

Receiver Oriented Variables (cont)

Variable Name:	GainAdjustMode
Description: **This value sets the mode of the input gain channels. 0-low gain, 1- high gain, 2- auto gain**	
Genre:	Receiver
Data Type:	int
Permissions:	Read/Write
Val :	0-low ; 1-high ; 2-automatic mode
Syntax **:** **{ "Command": "GetValue", "Arguments": "GainAdjustMode int 0"}** **{ "Command": "SetValue", "Arguments": " GainAdjustMode int 2 0"}**	
Return : **{" GainAdjustMode":value}**	

Variable Name:	InbandEnergy
Description: **This parameter is the smoothed averaged sum of squares of the input passband energy.**	
Genre:	Receiver
Data Type:	float
Permissions:	Read
Min :	0.
Max :	Max float
Syntax **:** **{ "Command": "GetValue", "Arguments": "InbandEnergy float 0"}**	
Return : **{" InbandEnergy":value}**	

Receiver Oriented Variables (cont)

Variable Name:	InBandNoiseEnergy
Description: **This parameter is the smoothed averaged sum of squares of the input baseband energy sampled immediately prior to reception of a valid hopped acquisition signal.**	
Genre:	Receiver
Data Type:	float
Permissions:	Read
Min :	0.
Max :	Max float
Syntax **:** **{ "Command": "GetValue", "Arguments": "InBandNoiseEnergy float 0"}**	
Return : **{"InBandNoiseEnergy":value}**	

Variable Name:	LocalID
Description: **This value contains the ID number of the modem. Valid numbers are 0-255 (note 255 is designated as a broadcast ID)**	
Genre:	Receiver
Data Type:	int
Permissions:	Read/Write
Val :	0 - 254
	255 -broadcast
Syntax **:** **{ "Command": "GetValue", "Arguments": "LocalID int 0"}** **{ "Command": "SetValue", "Arguments": "LocalID int 127 0"}**	
Return : **{"GainAdjustMode":value}**	

Receiver Oriented Variables (cont)

Variable Name:	MODEM_Enable
Description: **This value enables modem processing. When disabled, transmitter and receiver will not process, but recording and playout still operate.**	
Genre:	Receiver/Transmit
Data Type:	int
Permissions:	Read/Write
Val :	0- modem disable 1-modem enable
Syntax **:** **{ "Command": "GetValue", "Arguments": "MODEM_Enable int 0"}** **{ "Command": "SetValue", "Arguments": "MODEM_Enable int 1 0"}**	
Return : **{"MODEM_Enable":value}**	

Variable Name:	PSK_BnTaps
Description: **This value contains the number of backwards taps for the PSK equalizer. Note that the number of backwards taps + forward_taps is constrained to be less than 70 taps. The default value is 6**	
Genre:	Receiver/Transmit
Data Type:	int
Permissions:	Read/Write
Min :	0
Max :	<70 NumFwd+NumBwd
Syntax **:** **{ "Command": "GetValue", "Arguments": "PSK_BnTaps int 0"}** **{ "Command": "SetValue", "Arguments": "PSK_BnTaps int 8 0"}**	
Return : **{" PSK_BnTaps":value}**	

Receiver Oriented Variables (cont)

Variable Name:	PSK_Constellation
Description: **This value retrieves the 64 latest PSK constellation points**	
Genre:	Receiver
Data Type:	Float
Permissions:	Read
Min:	Min float
Max :	Max float
Syntax **:** **{ "Command": "GetValue", "Arguments": "PSK_Constellation float 0"}**	
Return : **{" PSK_Constellation":[v0,v1,v2,v3,…v63]}**	

Variable Name:	PSK_FnTaps
Description: **This value contains the number of forward taps for the PSK equalizer. Note that the number of backwards taps + forward_taps is constrained to be less than 70 taps. The default value is 44**	
Genre:	Receiver/Transmit
Data Type:	int
Permissions:	Read/Write
Min :	0
Max :	NumFwd+NumBwd < 70
Syntax **:** **{ "Command": "GetValue", "Arguments": "PSK_FnTaps int 0"}** **{ "Command": "SetValue", "Arguments": "PSK_FnTaps int 32 0"}**	
Return : **{" PSK_FnTaps":value}**	

Receiver Oriented Variables (cont)

Variable Name:	PSK_PDSNR
Description: **This parameter post detection SNR when in PSK modem**	
Genre:	Receiver
Data Type:	float
Permissions:	Read
Min :	0.
Max :	Max float
Syntax **:** **{ "Command": "GetValue", "Arguments": "PSK_PDSNR float 0"}**	
Return : **{"PSK_PDSNR":value}**	

Variable Name:	PSK_PLL
Description: **This value contains error measured within the PLL when in PSK mode.**	
Genre:	Receiver
Data Type:	float
Permissions:	Read
Min:	0
Max :	MAX float
Syntax **:** **{ "Command": "GetValue", "Arguments": "PSK_PLL float 0"}**	
Return : **{"PSK_PLL":value}**	

Receiver Oriented Variables (cont)

Variable Name:	PSK_Taps
Description: **This value retrieves the backward and forward equalizer taps. Note they are concatenated backward + forward.**	
Genre:	Receiver
Data Type:	Float
Permissions:	Read
Min:	Min float
Max :	Max float
Syntax **:** **{ "Command": "GetValue", "Arguments": "PSK_Taps float 0"}**	
Return : **{" PSK_Taps":[b0, b1, b2, bn, f0, f1,…fm]}** **n backward taps, m forward taps**	

Variable Name:	RangeTimeout_mS
Description: **This value contains the fixed turnaround delay from received range request to transmit range response. This delay is measured in mS. The default value is 15000**	
Genre:	Receiver/Transmit
Data Type:	int
Permissions:	Read/Write
Min :	0
Max :	60000
Syntax **:** **{ "Command": "GetValue", "Arguments": "PSK_FnTaps int 0"}** **{ "Command": "SetValue", "Arguments": "PSK_FnTaps int 32 0"}**	
Return : **{"RangeTimeout_mS":value}**	

Receiver Oriented Variables (cont)

Variable Name:	RxEnable
Description: **This value is used to enable/disable the receiver processing.**	
Genre:	Receiver
Data Type:	int
Permissions:	Read/Write
Min :	0- receiver disable
Max :	1- receiver enable
Syntax **:** **{ "Command": "GetValue", "Arguments": "RxEnable int 0"}** **{ "Command": "SetValue", "Arguments": "RxEnable int 1 0"}**	
Return : **{" RxEnable":value}**	

Variable Name:	RxScramblerMode
Description: **This value enables or disables the data scrambler for PSK payloads**	
Genre:	Receiver/Transmit
Data Type:	int
Permissions:	Read/Write
Min :	0- disable PSK payload scrambler
Max :	1- enable PSK payload scrambler
Syntax **:** **{ "Command": "GetValue", "Arguments": "RxScramblerMode int 0"}** **{ "Command": "SetValue", "Arguments": "RxScramblerMode int 1 0"}**	
Return : **{" RxScramblerMode":value}**	

Receiver Oriented Variables (cont)

Variable Name:	rxState
Description: **This value is used to enable/disable the analog board receiver board hardware. It is not recommended for users to modify.**	
Genre:	Receiver
Data Type:	int
Permissions:	Read/Write
Min :	0- receiver hardware disable
Max :	1- receiver hardware enable
Syntax **:** **{ "Command": "GetValue", "Arguments": "rxState int 0"}** **{ "Command": "SetValue", "Arguments": "rxState int 1 0"}**	
Return : **{"rxState":value}**	

Variable Name:	SignalEnergy
Description: **This parameter is the smoothed averaged sum of squares of the input baseband energy sampled during reception of a valid hopped acquisition signal.**	
Genre:	Receiver
Data Type:	float
Permissions:	Read
Min :	0.
Max :	Max float
Syntax **:** **{ "Command": "GetValue", "Arguments": "SignalEnergy float 0"}**	
Return : **{"SignalEnergy":value}**	

Receiver Oriented Variables (cont)

Variable Name:	SNR
Description: **This parameter is the ratio of signal to noise expressed in Db and captured during a valid acquisition.**	
Genre:	Receiver
Data Type:	float
Permissions:	Read
Min :	0.
Max :	Max float
Syntax **:** **{ "Command": "GetValue", "Arguments": "SNR float 0"}**	
Return : **{“SNR”:value}**	

Variable Name:	SSB_SqLevel
Description: **This value sets the voice received squelch level. A level of zero corresponds to no squelch. This value is set by a user to just above the background noise to mute the receiver until such time as a signal arrives above the noise.**	
Genre:	Receiver
Data Type:	float
Permissions:	Read/Write
Min :	0.
Max :	1.
Syntax **:** **{ "Command": "GetValue", "Arguments": "SSB_SqLevel float 0"}** **{ "Command": "SetValue", "Arguments": "SSB_SqLevel float .5 0"}**	
Return : **{“SSB_SqLevel”:value}**	

Receiver Oriented Variables (cont)

Variable Name:	SSB_Volume
Description: **This value sets the voice received volume level**	
Genre:	Receiver
Data Type:	float
Permissions:	Read/Write
Min :	0.
Max :	100.
Syntax **:** **{ "Command": "GetValue", "Arguments": "SSB_Volume float 0"}** **{ "Command": "SetValue", "Arguments": "SSB_Volume float 10 0"}**	
Return : **{“SSB_Volume”:value}**	

Transmitter Oriented Variables

Variable Name:	CarrierTxMode
Description: **This value enables or disables the carrier only transmit mode. In this mode, regardless of the data sent to the modem, a single tone at the carrier frequency is transmitted. This mode is useful for debug and experimentation.**	
Genre:	Transmitter
Data Type:	int
Permissions:	Read/Write
Min :	0- (default) modem mode
Max :	1- tone mode
Syntax **:** **{ "Command": "GetValue", "Arguments": "CarrierTxMode int 0"}** **{ "Command": "SetValue", "Arguments": "CarrierTxMode int 1 0"}**	
Return : **{"CarrierTxMode":value}**	

Variable Name:	ConsolePacketBytes
Description: **This value corresponds to the number of bytes received on the telnet console to trigger an autosend of the telenet data. This parameter is intended to make console to console operation easier to use.**	
Genre:	Transmitter
Data Type:	int
Permissions:	Read/Write
Min :	1
Max :	8192
Syntax **:** **{ "Command": "GetValue", "Arguments": "ConsolePacketBytes int 0"}** **{ "Command": "SetValue", "Arguments": "ConsolePacketBytes int 256 0"}**	
Return : **{"ConsolePacketBytes":value}**	

Transmitter Oriented Variables (cont)

Variable Name:	ConsoleTimeoutMS
Description: **This value corresponds to time in milliseconds for the telnet console to trigger an autosend of the telenet data when data is available. This parameter is intended to make console to console operation easier to use.**	
Genre:	Transmitter
Data Type:	int
Permissions:	Read/Write
Min :	0
Max :	60000
Syntax **:** **{ "Command": "GetValue", "Arguments": "ConsoleTimeoutMS int 0"}** **{ "Command": "SetValue", "Arguments": "ConsoleTimeoutMS int 10000 0"}**	
Return : **{"ConsoleTimeoutMS":value}**	

Variable Name:	PayloadMode
Description: **This value corresponds modulation data rate of the Payload portion of the waveform.**	
Genre:	Transmitter
Data Type:	int
Permissions:	Read/Write
Values :	0- 80bps Frequency Hop
	1- 5120 bps PSK
	2- 2560 bps PSK
	3- 1280 bps PSK
	4- 640 bps PSK
	5- 10240 bps (uncoded) PSK
Syntax **:** **{ "Command": "GetValue", "Arguments": "PayloadMode int 0"}** **{ "Command": "SetValue", "Arguments": "PayloadMode int 1 0"}**	

Return : **{“ PayloadMode”:value}**

Transmitter Oriented Variables (cont)

Variable Name:	PeakEnvelopePower
Description: **This value contains the peak envelope power of the previous SSB transmission. Reading this variable clears the value.**	
Genre:	Transmitter
Data Type:	float
Permissions:	Read
Min :	0.
Max :	MAX float
Syntax **:** **{ "Command": "GetValue", "Arguments": "PeakEnvelopePower float 0"}**	
Return : **{“PeakEnvelopePower”:value}**	

Variable Name:	PlayMode
Description: **This value corresponds places the transmitter waveform player in either passband mode or baseband mode.**	
Genre:	Transmitter
Data Type:	int
Permissions:	Read/Write
Min :	0- passband mode
Max :	1- baseband mode
Syntax **:** **{ "Command": "GetValue", "Arguments": "PlayMode int 0"}** **{ "Command": "SetValue", "Arguments": "PlayMode int 1 0"}**	
Return : **{“PlayMode”:value}**	

Transmitter Oriented Variables (cont)

Variable Name:	RemoteID
Description: **This value contains the intended receiver ID for the transmission.**	
Genre:	Transmitter
Data Type:	int
Permissions:	Read/Write
Min :	0
Max :	254 255 (broadcast)
Syntax **:** **{ "Command": "GetValue", "Arguments": "RemoteID int 0"}** **{ "Command": "SetValue", "Arguments": "RemoteID int 12 0"}**	
Return : **{"RemoteID":value}**	

Variable Name:	SSB_Txpower
Description: **This value contains a scalar factor that scales the transmit output prior to transmission.**	
Genre:	Transmitter
Data Type:	float
Permissions:	Read/Write
Min :	0.
Max :	10.
Syntax **:** **{ "Command": "GetValue", "Arguments": "SSB_Txpower float 0"}** **{ "Command": "SetValue", "Arguments": "SSB_Txpower float 0.3 0"}**	
Return : **{"SSB_Txpower":value}**	

Transmitter Oriented Variables (cont)

Variable Name:	SSB_VxLevel
Description: **This value contains a scalar that sets the level for a voice activated transmit switch when in voice mode and SSB_VxMode is enabled. This input voice level must be exceeded for the transmitter to self enable, once enabled the transmitter remains on until a prescribed hangover period of silence expires. Note the default value is .005.**	
Genre:	Transmitter
Data Type:	float
Permissions:	Read/Write
Min :	0.
Max :	1.
Syntax **:** **{ "Command": "GetValue", "Arguments": "SSB_ VxLevel float 0"}** **{ "Command": "SetValue", "Arguments": "SSB_ VxLevel float 0.003 0"}**	
Return : **{"SSB_ VxLevel":value}**	

Variable Name:	StreamingTxLen
Description: **This value contains the number of bytes to be set when uploading a file.**	
Genre:	Transmitter
Data Type:	int
Permissions:	Read/Write
Min :	1
Max :	8192
Syntax **:** **{ "Command": "GetValue", "Arguments": "StreamingTxLen int 0"}** **{ "Command": "SetValue", "Arguments": "StreamingTxLen int 1045 0"}**	
Return : **{"StreamingTxLen":value}**	

Transmitter Oriented Variables (cont)

Variable Name:	tpaState
Description: **This value is an enable variable that is used to powerup or powerdown the transimit final amplifier. It is not recommended for typical users to modify this variable.**	
Genre:	Transmitter
Data Type:	int
Permissions:	Read/Write
Min :	0
Max :	1
Syntax **:** **{ "Command": "GetValue", "Arguments": "tpaState int 0"}** **{ "Command": "SetValue", "Arguments": "tpaState int 0 0"}**	
Return : **{"tpaState":value}**	

Variable Name:	TxChirpMode
Description: **This value is an enable flag for setting the transmitter into 'chirp mode.' In this mode the transmitter sends an lfm chirp followed by silence prior to sending the acquisition sequence. This mode is used for characterizing channels by transmitting a known signal and receiving the signal shaped by the channel.**	
Genre:	Transmitter
Data Type:	int
Permissions:	Read/Write
Min :	0- disabled (default)
Max :	1- enabled
Syntax **:** **{ "Command": "GetValue", "Arguments": "TxChirpMode int 0"}** **{ "Command": "SetValue", "Arguments": "TxChirpMode int 1 0"}**	
Return : **{"TxChirpMode":value}**	

Transmitter Oriented Variables (cont)

Variable Name:	TxPowerWatts
Description: **This value contains a parameter for the desired power in watts for the transmission. It presumes the transducer has gone through a calibration phase for accurate operation.**	
Genre:	Transmitter
Data Type:	float
Permissions:	Read/Write
Min :	0.
Max :	100.
Syntax **:** **{ "Command": "GetValue", "Arguments": "TxPowerWatts float 0"}** **{ "Command": "SetValue", "Arguments": "TxPowerWatts float 30 0"}**	
Return : **{" TxPowerWatts":value}**	

Variable Name:	TxTimeout_mS
Description: **This value is the number of milliseconds that the transmitter can remain transmitting a packet before a timeout will occur. The timeout terminates the transmission state.**	
Genre:	Transmitter
Data Type:	int
Permissions:	Read/Write
Min :	0
Max :	6000000 {default}
Syntax **:** **{ "Command": "GetValue", "Arguments": "TxTimeout_mS int 0"}** **{ "Command": "SetValue", "Arguments": "TxTimeout_mS int 15000 0"}**	
Return : **{"TxTimeout_mS":value}**	

Variable Name:	UPCONVERT_Carrier
Description: **This value sets the transmitter upconverter carrier in Hz.**	
Genre:	Transmitter
Data Type:	int
Permissions:	Read/Write
Min :	20000
Max :	59750
Syntax **:** **{ "Command": "GetValue", "Arguments": "UPCONVERT _Carrier int 0"}** **{ "Command": "SetValue", "Arguments": "UPCONVERT _Carrier int 25000 0"}**	
Return : **{"UPCONVERT _Carrier":value}**	

Variable Name:	UPCONVERT_OutputScale
Description: **This value sets the transmitter upconverter carrier in Hz.** **Note setting the transmitter power via the API will be overwritten if this message is sent.**	
Genre:	Transmitter
Data Type:	float
Permissions:	Read/Write
Min :	0.
Max :	10.
Syntax **:** **{ "Command": "GetValue", "Arguments": "UPCONVERT _ OutputScale float 0"}** **{ "Command": "SetValue", "Arguments": "UPCONVERT _ OutputScale float 1. 0"}**	
Return : **{"UPCONVERT _ OutputScale":value}**	

Variable Name:	RNG_TA_DelayMs
Description: **This value is the number of milliseconds that the transmitter waits from reception of a range request. The default value is 5000**	
Genre:	Ranging
Data Type:	int
Permissions:	Read/Write
Min :	3000
Max :	8000
Syntax **:** **{ "Command": "GetValue", "Arguments": "RNG_TA_DelayMs int 0"}** **{ "Command": "SetValue", "Arguments": "RNG_TA_DelayMs 7000 0"}**	
Return : **{"RNG_TA_DelayMs":value}**	

Popoto Python API Reference

Importing the python API

To access all of the functionality of the popoto API in Python it is necessary to import the API. This is done by using the Python **import** command at the start of your Python application. This is illustrated in the example below.

Connecting to the Socket

Connection to Popoto is done using standard IP methods of the high level language in use. For example:

```
# Import the Popoto modem API
import popoto
import socket
# Connect to Popoto Command Socket
cmdsocket=socket(AF_INET, SOCK_STREAM)
cmdsocket.connect(('10.0.0.233', 17000))
cmdsocket.settimeout(20)


# Issue Popoto Commands
```

Popoto Python API Reference

Popoto Python API

class popoto.popoto(ip, basePort)

This class can be run on the local Popoto Modem, or can be run remotely on a PC.

All commands are sent via function calls, and all JSON encoded responses and status from the modem are enqueued as Python objects in the reply queue.

In order to do this, the class launches a processing thread that looks for replies decodes the JSON and adds the resulting python object into the reply queue.

The Popoto class requires an IP address and port number to communicate with the Popoto Modem. This Port number corresponds to the base port of the modem application.

calibrateTransmit()

DESCRIPTION

calibrateTransmit send performs a calibration cycle on a new transducer to allow transmit power to be specified in watts. It does this by sending a known amplitude to the transducer while measuring voltage and current across the transducer. The resulting measured power is used to adjust scaling parameters in Popoto such that future pings can be specified in watts.

PARAMETERS

RESPONSES

SUCCESS

FAILURE

EXAMPLE

drainReplyQ()

This function reads and dumps any data that currently resides in the Popoto reply queue. This function is useful for putting the replyQ in a known empty state.

PARAMETERS

RESPONSES

SUCCESS

FAILURE

EXAMPLE

getAllParameters()

Gets all Popoto control element info strings for all elements.

PARAMETERS

RESPONSES

SUCCESS

FAILURE

EXAMPLE

getParameter(*idx*)

Gets a Popoto control element info string by element index.

Parameters **idx** (*number*) – The index is the reference number of the element

PARAMETERS

RESPONSES

SUCCESS

FAILURE

EXAMPLE

getParametersList()

Gets the parameters list from the system controller.

PARAMETERS

RESPONSES

SUCCESS

FAILURE

EXAMPLE

getRtc()

Gets the real time clock date and time.

Returns clockstr The clockstr contains the value of the date in string format YYYY.MM.DDHH:MM;SS

PARAMETERS

RESPONSES

SUCCESS

FAILURE

EXAMPLE

getValueF(*Element*)

Gets the 32bit floating value of a Popoto float variable

Parameters **Element** (*string*) – The name of the variable to be retrieved

Returns value The value

PARAMETERS

RESPONSES

SUCCESS

FAILURE

EXAMPLE

getValueI(*Element*)

Gets an integer value of a Popoto integer variable

Parameters **Element** (*string*) – The name of the variable to be retreived

Returns value the Element's value

PARAMETERS

RESPONSES

SUCCESS

FAILURE

EXAMPLE

getVersion()

Retrieve the software version of Popoto

PARAMETERS

RESPONSES

SUCCESS

FAILURE

EXAMPLE

playPcmLoop(*inFile*, *bb*)

Play passband/baseband PCM for duration seconds. :param inFile: In file :type inFile: string :param bb:

PARAMETERS

RESPONSES

SUCCESS

FAILURE

EXAMPLE

playStartTarget(*filename*, *scale*)

Play a PCM file of 32bit IEEE float values out the transmitter Playback is passband if Popoto 'PlayMode' is 0 Playback is baseband if Popoto 'PlayMode' is 1

Parameters

- **filename** (*string*) – The filename of the pcm file on the SD card
- **scale** (*number*) – The transmitter scale value 0-10; higher numbers result in higher transmit power.

PARAMETERS

RESPONSES

SUCCESS

FAILURE

EXAMPLE

playStopTarget()

End playout of stored PCM file through Popoto transmitter

PARAMETERS

RESPONSES

SUCCESS

FAILURE

EXAMPLE

recPcmLoop(*outFile*, *duration*, *bb*)

recPcmLoop records passband/baseband pcm for duration seconds. This function also returns a vector of timestamps in pcmCount and a vector of HiGain_LowGain flags 0=lo,1=hi which indicate which A/D channel was selected on a frame basis

Code sets baseband mode as selected on input, but changes back to pass band mode on exit. Base band recording and normal modem function are mutually exclusive, as they share the Modem's Digital up converter.

Parameters

- **outFile** (*string*) – The output filename with path
- **duration** (*number*) – The duration of recording in seconds
- **bb** (*number 0/1 passband/baseband*) – passband or baseband selection

PARAMETERS

RESPONSES

SUCCESS

FAILURE

EXAMPLE

recordStartTarget(*filename*, *duration*)

Initiate recording acoustic signal data to the local SD card. Recording is passband if Popoto 'RecordMode' is 0 Recording is baseband if Popoto 'RecordMode' is 1

Parameters

- **filename** (*string*) – The filename on the local filesystem with path
- **duration** (*number*) – The duration in seconds for continuous record to split-up files with autonaming. Typical value is 60 for 1 minute files.

PARAMETERS

RESPONSES

SUCCESS

FAILURE

EXAMPLE

recordStopTarget()

Turn off recording to local SD card

PARAMETERS

RESPONSES

SUCCESS

FAILURE

EXAMPLE

send(*message*)

The send function is used to send a command with optional arguments to Popoto as a JSON string

Parameters **message** (*string*) – The message contains a Popoto command with optional arguments

PARAMETERS

RESPONSES

SUCCESS

FAILURE

EXAMPLE

sendRange(*power=0.1*)

Send a range request from a local Popoto Modem to initiate a ranging cycle to another remote Popoto Modem

PARAMETERS

power (floating point *number*) – The power in watts that is used for the entire ranging cycle. The remote modem will respond with a power that matches the requesting modem.

RESPONSES

SUCCESS

Upon a successful ranging cycle the local modem will report a range message as follows:

{"Range":<distance>,"Roundtrip Delay":<one way delay>,"SpeedOfSound":1500.000000,"Units":"m, ms, meters per second"}

FAILURE

If no reply is received within the configured range time window, the modem will return a timeout alert as follows:

{"Alert":"Timeout"}

EXAMPLE

```
ReplyTimeout=20 # This is the reply timeout in seconds
remoteID=33 # Set the value of the remote modem we are ranging to
Popoto.setvalueI('RemoteID', remoteID)
Popoto.sendRange(10.) # Send ranging at a power of 10 watts
print(Popoto.waitForReply(ReplyTimeOut)) # print the response
```

OUTPUT

```
{"Range":235.964355,"Roundtrip
Delay":314.619141,"SpeedOfSound":1500.000000,"Units":"m, ms, meters
per second"}
```

setRtc(*clockstr*)

Sets the real time clock.

Parameters **clockstr** (*string*) – The clockstr contains the value of the date in string format *YYYY.MM.DD-HH:MM;SS*

Note: there is no error checking on the string so make it right

PARAMETERS

RESPONSES

SUCCESS

FAILURE

EXAMPLE

setValueF(*Element*, *value*)

Sets a 32bit float value of a Popoto float variable

Parameters

- **Element** (*string*) – The name of the variable to be set
- **value** (*float*) – The value

PARAMETERS

RESPONSES

SUCCESS

FAILURE

EXAMPLE

setValueI(*Element*, *value*)

Sets an integer value of a Popoto integer variable

Parameters

- **Element** (*string*) – The name of the variable to be set
- **value** (*integer*) – The value

PARAMETERS

RESPONSES

SUCCESS

FAILURE

EXAMPLE

startRx()

startRx places Popoto modem in receive mode.

PARAMETERS

RESPONSES

SUCCESS

FAILURE

EXAMPLE

streamUpload(*filename*, *power*)

streamUpload Uploads a file for acoustic transmission

Parameters

- **filename** (*string*) – The filename to be sent with path
- **power** (*number*) – The desired power in watts

PARAMETERS

RESPONSES

SUCCESS

FAILURE

EXAMPLE

tearDownPopoto()

The tearDownPopoto method provides a graceful exit from any python Popoto script

PARAMETERS

RESPONSES

SUCCESS

FAILURE

EXAMPLE

transmitJSON(*JSmessage*)

The transmitJSON method sends an arbitrary user JSON message for transmission out the acoustic modem.

Parameters **JSmessage** (*string*) – The Users JSON message

PARAMETERS

RESPONSES

SUCCESS

FAILURE

EXAMPLE

waitForReply(*Timeout*)

waitForReply is a method that blocks on the replyQ until either a reply has been received or a timeout (in seconds) occurs.

Parameters **Timeout** (*{ type_description }*) – The timeout

PARAMETERS

RESPONSES

SUCCESS

FAILURE

EXAMPLE

Connecting Matlab™ to Popoto

In Matlab™ navigate to the API/Matlab directory. Let us assume the ip address of your popoto modem is 10.0.0.151. Then to connect simply issue the following command:

```
>> p=popoto('10.0.0.151',18500,0)
```

Matlab™ responds as follows:

```
popoto with properties:

               cmdsocket: [1×1 tcpclient]
            pcmlogsocket: []
             pcmiosocket: [1×1 tcpclient]
              datasocket: [1×1 tcpclient]
                   sport: []
              dispatcher: {9×2 cell}
          validPacketCnt: 0
               pingTimer: 0
              TxComplete: 0
                 carrier: 30000
                SampFreq: 102400
               Framesize: 640
                 cmdport: 18500
                dataport: 18501
             pcmlogport: 18502
              pcmioport: 18503
                  range: 0
                     ip: '10.0.0.151'
                dspmode: 0
               errorCnt: 0
          validHeaderCnt: 0
        constellationPts: []
                fwd_taps: []
                bwd_taps: []
              fwd_taplen: 0
              bwd_taplen: 0
             pll_theta_p: []
             pll_theta_w: []
                QCounter: 0
                    mips: []
              mipslegend: []
```

Your handle now is a variable called p. You now have access to all the variables and methods to control your popoto from Matlab™. Note that the tab completion feature of Matlab™ is extremely useful here. Simply type p. at the Matlab™ command prompt after a successful connection and Matlab™ will bring up a dialogbox of all methods.

Method Summary

RecPcmLoop	Record passband pcm for duration seconds. This function also
SetPlayMode	this function sets the modem in BaseBand or PassBand

SetRecordMode	this function sets the modem in BaseBand Recording
calibrateTransmit	send performs a calibration cycle on a new transducer
drainReplyQ	Read all bytes out of the command socket and dump them
getAllParameters	Gets all Popoto control element info strings for all elements.
getParameter	Gets a Popoto control element info string by element index.
getRtc	This function queries Popoto real time clock
getValueF	This function queries a floating point value to a supported popoto
getValueI	This function queries in integer value to a supported popoto
getVersion	This function queries Popoto software version
playPcmLoop	
playStartTarget	Play a PCM file of 32bit IEEE float values out the transmitter
playStopTarget	End playout of stored PCM file through Popoto transmitter
recordStartTarget	This function turns on the pcm recorder on the modem hardware
recordStopTarget	This function turns off the pcm recorder on the modem hardware
send	This functions sends a command string to the popoto
sendRange	This function instructs the modem TX to send a ranging
setRtc	This function sets Popoto real time clock
setValueF	This function sets a float value to a supported popoto
setValueI	This function sets an integer value to a supported popoto
startRx	This function turns on the popoto receiver
streamUpload	Upload a file for acoustic transmission
tearDownPopoto	Tear Down Popoto Object

transmitJSON	Send an arbitrary JSON message out the Popoto acoustic
waitForReply	Wait for data up to timeout period in the command socket