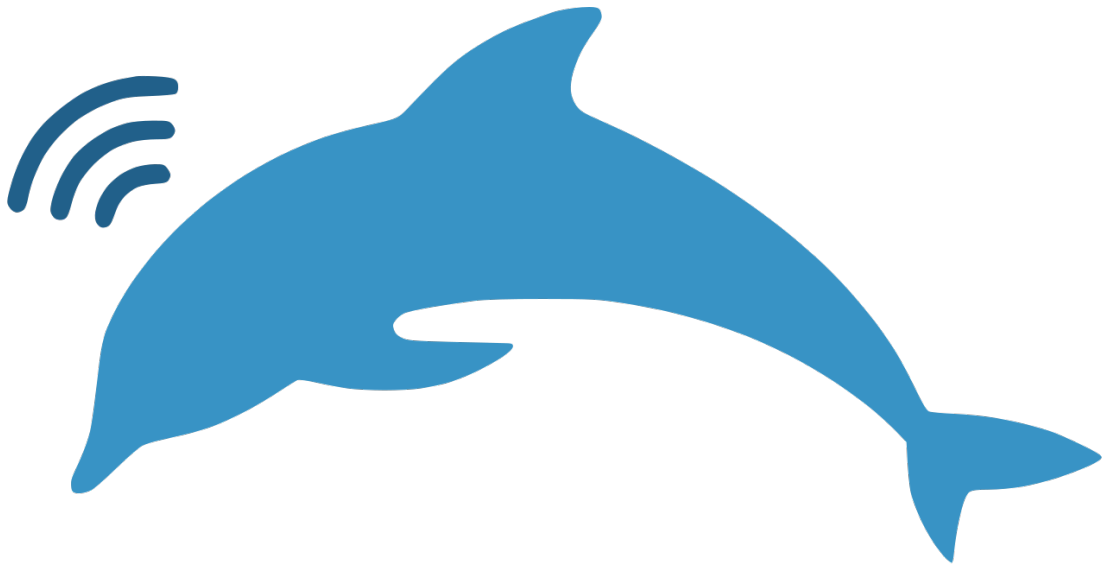

PopotoModem

Popoto Python API Reference

Feb 06, 2026

Contents

Welcome to the Popoto Modem Python API Reference Manual. This documentation is generated from the package's docstrings and provides both narrative guides and API reference material.

1 Overview

1.1 Introduction

At a low level, the Popoto Modem is controlled by JSON API messages sent over a TCP socket. These API messages provide a human readable format which may be used to design custom control software for the Popoto Modem. The Python API provides an abstraction of these control messages, and can be used for easy integration with custom Python scripts and programs.

1.2 High-level Description of the Popoto API

1.2.1 Sockets

The socket-based I/O structure of the Popoto software provides a flexible framework for interface, test, and portability. With a small amount of additional code, these sockets can be used to control the modem over standard interfaces such as RS-422, RS-232, and Ethernet. By default, the TCP Sockets on the modem use the following ports to communicate with the API:

Port	Role	Notes
17000	Commands	
17001	Data	
17002	PCM Logging	Not For Typical Use
17003	PCM Streaming	Not For Typical Use
17004	PCM Playback	Not For Typical Use

The Python API primarily uses its connection to the Command Port (17000) to control the modem. It connects to the modem's IP address and the appropriate port using Python's socket module.

1.2.2 JSON Command Messages

As stated above, the messages used to directly control the Popoto Modem are in a JSON format. This has two main advantages:

1. The messages are human-readable, and thus easy to understand at a glance.
2. Libraries for parsing JSON are available in most major programming languages, allowing for integration with a large variety of applications.

1.2.3 Commands

Commands sent to the Popoto Modem are represented as specially-formatted JSON objects. These consist of two parts, the Command keyword, which represents the action to be performed, and the Argument keyword, which represents the parameters with which to perform the action. The basic structure of a Popoto JSON command is as follows:

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

Note

Getting the software version

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

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

The modem responds with its version number in the following format:

```
{"Info":"Popoto Modem Version 4.7.0-beta+950.f8fbd.a5f0345c2"}
```

1.2.4 System-Level Variables

The Popoto Modem contains various variables/parameters which can be used to adjust the modem's setup or provide the user with information regarding the state of the modem.

Note

Setting the Carrier Frequency

One simple variable which can be used to demonstrate variable get/set capabilities is the Carrier variable, which dictates the modem's carrier frequency. In this example, the carrier is set to 25 kHz.

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

This would result in the modem responding as follows:

```
{"Info":"Value Set Carrier=25000"}
```

1.3 The Python Layer

In addition to direct use of the JSON API, the modem can be controlled using its Python API. This is installable via pip from the Popoto Modem repository on PyPi (`pip install popoto-api`), or directly using the popoto.py file, which is available on GitLab.

The Python API command reference can be found in a subsequent section of this document.

1.3.1 Example Usage

The example below configures two modems, sends a short JSON payload from the transmitter, and waits for the receiver to return the decoded data.

```
#!/usr/bin/env python3
from popoto import popoto

TX_IP, RX_IP = "10.0.0.121", "10.0.0.122"
MSG = "Hello World"

rx = popoto(RX_IP)
tx = popoto(TX_IP)

tx.set("TxPowerWatts", 1)
tx.set("Carrier", 30000)
rx.set("Carrier", 30000)
tx.transmitJSON('{"Payload":{"Data":"' + MSG + '"} }')

print(
    rx.waitForSpecificReply("Data", list(bytes(MSG, "ASCII")))
)

tx.tearDownPopoto()
rx.tearDownPopoto()
```

1.4 Pshell

For direct, user-level access, the pshell can be used, which provides a helpful CLI for controlling the modem. The pshell is implemented in both python (default) and rust (available on request).

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

2 API Reference

2.1 Popoto Client

class popoto_api.popoto.**popoto**(*ip='localhost'*, *basePort=17000*, *logname=None*)

Bases: object

High-level client for controlling a Popoto Modem over TCP.

Public methods issue commands to the Popoto application, which the modem responds to with JSON strings. These strings are collected and parsed by a background worker thread that enqueues the resulting Python dictionaries in the *popoto* instance's reply queue (*popoto.replyQ*).

Initialize the class with the modem's host address and base application port. The base port corresponds to the control port exposed by the modem firmware – by default, this is TCP port 17000.

Constructor Args:

- **ip (str):**
  Hostname or IP address of the target Popoto modem.
- **basePort (int):**
  Base TCP port exposed by the modem application; other service ports are derived from this value.
- **logname (str | None):**
  Optional logger name to configure the instance-specific logger.

logger

Instance-specific logger.

Type
logging.Logger

pcmplayport

PCM playback port derived from *basePort*.

Type
int

pcmioport

PCM I/O port derived from *basePort*.

Type
int

pcmlogport
PCM log port derived from *basePort*.
Type
int
dataport
Data port derived from *basePort*.
Type
int
cmdport
Control port derived from *basePort*.
Type
int
rawTerminal
Toggle for ANSI styling in displayed replies.
Type
bool
quiet
Reply display verbosity flag used by helper methods.
Type
int
cmdsocket
Command socket to the modem.
Type
socket.socket
SampFreq
Sample frequency for audio streaming.
Type
int
pcmplaysocket
PCM playback socket (0 until opened).
Type
socket.socket | int
recByteCount
Byte counter for PCM receive.
Type
int
ip
Hostname or IP address of the target modem.
Type
str
is_running
Indicates whether the client is running.
Type
bool

fp

Open file handle for PCM I/O operations.

Type

io.IOBase | None

fileLock

Lock guarding file I/O.

Type

threading.Lock

rxThread

Background command receive thread.

Type

threading.Thread

replyQ

Queue of parsed JSON replies.

Type

queue.Queue

datasocket

Data socket for payloads.

Type

socket.socket | None

intParams

Cached integer parameter values.

Type

dict[str, int]

floatParams

Cached float parameter values.

Type

dict[str, float]

paramsList

Parameter metadata list.

Type

list[dict]

varcache

General-purpose cached values.

Type

dict[str, object]

cachingEnabled

Toggle for parameter caching.

Type

bool

isRemoteCmd

True when running a remote command.

Type
bool

remoteCommandAck
Tracks remote command acknowledgements.
Type
int | bool

remotePshellEnabled
True when remote pshell is enabled.
Type
bool

remotePshellCommandQueue
Queue for remote pshell commands.
Type
queue.Queue

remoteCommandHandler
Handler for remote command payloads.
Type
object | None

CurrentCommandRemote
True if current command is remote.
Type
bool

pcmiosocket
PCM I/O socket.
Type
socket.socket | None

MapPayloadModesToRates
Payload mode to bitrate map (bps).
Type
list[int]

first_recording
True if first PCM recording session.
Type
bool

running_on_modem
True when running on modem hardware.
Type
bool

Note

Examples:
Initialize and connect to the popoto_app on localhost:17000

```
p = popoto.popoto('localhost')
Initialize and connect to the popoto_app on 10.0.1.193:18000
p = popoto.popoto('10.0.1.193', 18000)
```

BitRateToPayloadMode(*rate*)

Return the payload mode index that matches a bitrate.

Parameters

rate (*int*) – Bitrate in bits per second.

Returns

Payload mode index.

Return type

int

Note

As of writing, the map of payload mode to bitrate is shared in *payLoadModeToBitRate()*. This is subject to change and new payload modes may be added.

ROBOT_LIBRARY_SCOPE = 'GLOBAL'

RxCmdLoop()

Continuously read and decode newline-delimited JSON messages from the command socket, enqueueing replies and handling reconnects as needed.

calibrateTransmit()

(Deprecated)

check_running_on_modem()

Determine whether the client is executing on Popoto modem hardware by checking CPU characteristics and the serial number marker file.

close()

Close the command socket connection.

dispMips(*mips*)

Pretty-print cycle statistics returned by `getCycleCount`.

Parameters

mips (*Mapping[str, Mapping[str, float]]*) – Raw statistics keyed by module name.

drainReplyQ()

Empty the reply queue while logging every dequeued message.

drainReplyQquiet()

Empty the reply queue without logging the removed messages.

exit()

Hook for compatibility with legacy client code.

expandCapturesPartition()

(Deprecated) Expand the captures partition to occupy remaining device space.

get(*Element*)

Request the value of a Popoto internal variable.

Parameters

Element (*str*) – Name of the internal variable to read.

getAllParameters()

Query the modem for metadata describing every internal variable.

getCycleCount()

Request and display application cycle-count statistics from the modem.

getExclusiveAccess()

Acquire exclusive access to the modem command socket.
Sets a mutex token; if another client holds the token, this method waits and retries until ownership is granted.

getParameter(*idx*)

Request metadata for an internal variable by index.

Parameters

idx (*int*) – Element index to query.

getParametersList()

Return the cached parameter list from the system controller.

getPopotoModulations()

request a json message from the modem that contains the supported modulations

getRecordingStatus()

Request the status and key information of the current recording session, in JSON Format.

getRtc()

Request the current real-time clock string from the modem.

getValueF(*Element*)

Request the value of a Popoto internal variable.

Parameters

Element (*str*) – Name of the internal variable to read.

getValueI(*Element*)

Request the value of a Popoto internal variable.

Parameters

Element (*str*) – Name of the internal variable to read.

getVersion()

Retrieve Popoto application and component versions from the connected modem.

mariaCommand(*JSmessage*)

MARIA command builder with automatic type parsing. If there is exactly

one argument, “Arguments” will be that item directly. If more than one, “Arguments” will be a list.

mountCapturesPartition()

(Deprecated) Mount `/dev/mmcblk0p3` at `/captures` if available.

payLoadModeToBitRate(*payloadMode*)

Return the bitrate associated with a payload mode.

Parameters

payloadMode (*int*) – Payload mode index.

Returns

Bitrate in bits per second.

Return type

int

Note

As of writing, the map of payload mode to bitrate is:

Rate	PayloadMode value
80 bps Frequency-Hopped (Default)	0
5120 bps QPSK	1
2560 bps QPSK	2
1920 bps QPSK	3
1280 bps QPSK	4
10240 bps Uncoded QPSK	5
2560 bps BPSK	6
1280 bps BPSK	7
960 bps BPSK	8
640 bps BPSK	9

This is subject to change and new payload modes may be added.

playPcmLoop(*inFile*, *scale*, *bb*)

Loop a REC/RAW file through the transmitter for playback testing.

Parameters

- **inFile** (*str*) – Path to the input REC/RAW file.
- **scale** (*float*) – Transmit scaling factor applied during playback.
- **bb** (*int*) – `0` for passband playback, `1` for baseband playback.

playStartTarget(*filename*, *scale*)

Play a stored WAV file through the Popoto transmitter.
Playback occurs in passband when `PlayMode` is `0` and baseband when it is `1`.

Parameters

- **filename** (*str*) – WAV file path (32-bit IEEE float samples).
- **scale** (*float*) – Transmitter scale factor (0-10) controlling output power.

playStopTarget()

Stop playback of the current WAV file.

prepareFilesystem()

Ensure capture storage is expanded and mounted on modem hardware.

pumpAudio(*inFilename*, *outFilename*)

Play audio through the modem DSP path and capture the processed output.

Parameters

- **inFilename** (*str*) – Input PCM/REC file path to send to the modem.
- **outFilename** (*str*) – Destination PCM file path for the returned audio.

pumpAudioDeprecated(*inFilename*, *outFilename*)

Legacy audio loopback helper retained for backwards compatibility.

Parameters

- **inFilename** (*str*) – Input PCM/REC file path to stream to the modem.
- **outFilename** (*str*) – Destination PCM file path for captured audio.

read_pcm(*sock*)

Read a single frame of floating-point PCM samples from the given socket.

Parameters

sock (*socket.socket*) – Connected PCM stream socket.

Returns

Array of `FRAMESIZE` float32 samples.

Return type

numpy.ndarray

reboot()

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

Record passband or baseband PCM data for a fixed duration.

> **Note**
>
> This function sets passband/baseband mode as specified, but changes back to passband mode on exit. Baseband recording and normal modem function are mutually exclusive, as they both require the use of modem's digital upconverter.

Parameters

- **outFile** (*str*) – Output REC/RAW file path.
- **duration** (*float*) – Recording duration in seconds.

- **bb** (*int*) – `0` for passband capture, `1` for baseband capture.

receive()

Read a raw chunk of 256 bytes from the command socket.

recordStartTarget(*filename*, *duration*, *capturesdir='/captures'*)

Begin recording acoustic data to the modem's internal storage or SD card.

- Recording is passband if Popoto 'RecordMode' is 0
- Recording is baseband if Popoto 'RecordMode' is 1

Parameters

- **filename** (*str*) – Destination file path; auto-prefixed with `capturesdir` when that directory exists.
- **duration** (*float*) – Duration in seconds for each capture file.
- **capturesdir** (*str*) – Directory used for capture storage when present.

recordStopTarget()

Stop recording modem audio.

register_message_handler(*message_type*, *callback*, *user_args*)

Register a callback for JSON replies that contain a given key. The callback will be called with two arguments: the intercepted message (as a dict) and the user_arg provided at registration time.

Parameters

- **message_type** (*str*) – JSON key that triggers the callback. For example, if you want to intercept messages that include the key "MyCustomType", pass that string here.
- **callback** (*Callable[[dict, Any], None]*) – Function invoked with the intercepted message and user argument.
- **user_arg** (*Any | None*) – Optional data passed through to *callback*.

register_reconnect_callback(*callback*, *user_args*)

Registers a callback function to The callback will be called with two arguments: the intercepted message (as a dict) and the user_arg provided at registration time.

Parameters

- **callback** – A function that will be called with user_arg as arguments.
- **user_arg** – Additional arguments that will be passed to the callback.

releaseExclusiveAccess()

Release the exclusivity token so other clients can issue commands.

send(*message*)

Send a Popoto JSON API command with optional arguments over the command socket.

Parameters

message (*str*) – Command string optionally followed by arguments.

sendPrecisionRange(*power=0.1*)

Initiate a precision ranging cycle to another modem.

> **Note**
>
> This is for precision ranging, which is mutually exclusive with range-with-data. To send a range request with data, use setRangeData() and sendRange().

Parameters

power (*float*) – Transmit power in watts.

sendRange(*power=0.1*)

Initiate a standard ranging cycle to another modem.

Parameters

power (*float*) – Transmit power in watts.

sendRemotePshell(*command*)

Issue a remote pshell command without requesting an acknowledgement.

> **Note**
>
> This function will send pshell commands, *NOT* JSON API calls. To send a JSON API command to a remote modem, use sendRemoteCommand.

Parameters

command (*str*) – Command string to execute on the remote shell.

sendRemotePshellAck(*command*)

Issue a remote pshell command and request an acknowledgement.

> **Note**
>
> This function will send pshell commands, *NOT* JSON API calls. To send a JSON API command to a remote modem, use sendRemoteCommand.

Parameters

command (*str*) – Command string to execute on the remote shell.

sendRemoteStatus(*message*)

Transmit a remote status message and await transmit completion.

Parameters

message (*str*) – Status payload to forward to the remote modem.

set(*Element*, *value*, *override_cache=False*)

Sets a value of a Popoto variable

Parameters

- **Element** (*str*) – Name of the internal variable to update.

- **value** (*Any*) – Value to apply to the variable.

setANSITerminal()

Enable ANSI styling for displayed modem replies.

setActiveChannels(*line*)

Enable JACK PCM streaming channels for recording.

> **Note**
>
> PMM6081 has 10 receiving channels, though this command can only set the mask to receive up to 8 of them at any given time. Two channels will always be ignored by this method; which channels these are depends on how you launch the popoto app.

Parameters

line (*str*) – Space-separated channel numbers or the string `"all"`.

setLocalCommand()

Route subsequent commands directly to the API's modem TCP connection. (Disables RemoteCmd)

setRangeData(*JSmessage*)

Configure the range data. This data serves as the payload to be sent along with a range request, as well as the response payload returned during ranging exchanges.

Parameters

- **JSmessage** (*str*) – JSON-encoded payload configuration message. May
- **data.** (*include a maximum of 255 integer-encoded bytes of*)

> **Note**
>
> The format of JSMessage is as follows:
> `{"Payload": {"Data": [<list of up to 255 bytes>]} }`
> For example:
> `{"Payload": {"Data": [72, 101, 108, 108, 111] } }`

setRawTerminal()

Disable ANSI styling for displayed modem replies.

setRemoteCommand(*AckFlag*)

Configure the JSON API to send commands to a remote modem acoustically.

> **Note**
>
> These commands will subsequently be executed on the remote modem. Not all commands can be executed remotely. This function will send JSON API calls, *NOT* pshell commands. To send a pshell command to a remote modem, use sendRemotePshell. The RemoteNode variable controls which modem to send the command to.

Parameters

- **AckFlag** (*bool*) – If True, request an acknowledgement from the remote
- **modem.**

setRemoteCommandHandler(*obj*)

Register a handler for remote command payloads from other modems.

Parameters

obj – Handler instance that implements `handleCommand(command)`.

setRtc(*clockstr*)

Set the modem real-time clock.

Parameters

clockstr (*str*) – Timestamp string in `YYYY.MM.DD-HH:MM;SS` format.

setTimeout(*timeout*)

Set the socket timeout used while waiting for command-channel traffic.

Parameters

timeout (*float*) – Timeout value in seconds.

setValueF(*Element*, *value*, *override_cache=False*)

Sets a value of a Popoto variable

Parameters

- **Element** (*str*) – Name of the internal variable to update.
- **value** (*Any*) – Value to apply to the variable.

setValueI(*Element*, *value*, *override_cache=False*)

Sets a value of a Popoto variable

Parameters

- **Element** (*str*) – Name of the internal variable to update.
- **value** (*Any*) – Value to apply to the variable.

startRx()

Put the Popoto modem into receive mode.

statReport(*line*)

(Deprecated) Log a status message locally and forward it over the remote command channel when operating in remote-command mode.

Parameters

line (*str*) – Human-readable status message.

streamBytes(*data*)

Chunk and transmit large payloads over the data socket, falling back to

command-socket JSON transmission for small payloads.

Parameters

data (*bytes | bytearray | io.BufferedReader | str*) – Payload to stream.

streamDownload(*filename*, *remotePowerLevel*)

(Deprecated) Download a file over the acoustic streaming channel. Issues a request for upload to the remote modem, then listens for a stream of data to download.

Parameters

- **filename** (*str*) – Destination filename; `.download` is appended to the provided base name.
- **remotePowerLevel** (*float | None*) – Optional power level to request from the remote modem before downloading.

streamUpload(*filename*, *power*, *PayloadMode=1*)

(Deprecated) Upload a file for acoustic transmission.

> **Note**
>
> This transmits the internal data of a file, it does not play back an audio file. For that functionality, use `playStartTarget()`.

Parameters

- **filename** (*str*) – Path to the file that should be transmitted.
- **power** (*float*) – Desired transmit power in watts.
- **PayloadMode** (*int*) – Payload mode index used for streaming (default
- **1)**.

tearDownPopoto()

Gracefully shut down an interactive Popoto session.
It is advised to call this when you no longer need a connection to the popoto application, or at the end of your python script.

transmitJSON(*JSmessage*)

Send an arbitrary payload of bytes through the acoustic modem.

Parameters

JSmessage (*dict | str*) – JSON-encoded string or list of bytes to transmit. Large payloads are automatically streamed in chunks.

> **Note**
>
> Does not transmit arbitrary JSON; rather, the user specifies the data to be transmitted using a JSON string or JSON-decodable python dictionary object, with the following format:
> `{"Payload": {"Data": [<list of up to 255 bytes>]} }`

For example:
`{"Payload": {"Data": [72, 101, 108, 108, 111] } }`
If you would like to send JSON, encode it as bytes first, then send the array of bytes (integer encoded ASCII) using this command.

unregister_message_handler(*message_type*)

Remove the registered callback for a JSON message type.

Parameters

message_type (*str*) – JSON key whose callback should be removed.

unregister_reconnect_callback(*callback*, *user_args=()*)

Unregisters the callback function for a given JSON message type.

Parameters

message_type – The JSON key for which the callback should be removed.

property verbose

Verbosity of output, from 0-5:

- 0: CRITICAL
- 1: ERROR
- 2: WARNING
- 3: INFO
- 4: DEBUG
- 5: UNSET (all)

Logging level is 0 - 50, 0 being most verbose and 50 being least verbose. The *verbose* property is 0 - 5, with 0 being least verbose and 5 being most verbose. Thus popoto.verbose is treated as *(max(logging.getLevelNamesMapping().values()) // 10) - (self.logger.level // 10)*.

waitForReply(*Timeout=10*)

Block until a reply arrives or the timeout elapses.

Parameters

Timeout (*float*) – Maximum number of seconds to wait.

Returns

Reply dictionary or `{"Timeout": 0}` on timeout.

Return type

dict

waitForSpecificReply(*Msgtype*, *value=None*, *Timeout=10*)

Block until a reply with the requested key/value pair arrives.

Parameters

- **Msgtype** (*str*) – Reply dictionary key to match.
- **value** (*Any or None*) – Optional value (or substring for str replies) to match.
- **Timeout** (*float*) – Maximum number of seconds to wait.

Returns

Matching reply dictionary or `{"Timeout": 0}` on timeout.

Return type
dict

write_pcm(*sock*, *buf*)

Send a frame of floating-point PCM samples to the modem PCM socket.

Parameters

- **sock** (*socket.socket*) – Connected PCM stream socket.
- **buf** (*Sequence[float]*) – FRAMESIZE samples to transmit.

Appendix A: 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.

A.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)

A.2 Coding Structure

A.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.

A.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.

A.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.

A.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.

A.4 Payload Mode Summary

Table 1: PayloadMode Summary

Mode	Mod	Rate (bps)	FEC	Repetition	Recommended
FH_80bps	FH-FSK	80	inherent	n/a	Yes
BPSK_640bps	BPSK	640	1/2	4×	Yes
BPSK_960bps	BPSK	960	1/2	2.67×	Yes
BPSK_1280bps	BPSK	1280	1/2	2×	Yes
BPSK_2560bps	BPSK	2560	1/2	1×	Yes
QPSK_1280bps	QPSK	1280	1/2	4×	No
QPSK_1920bps	QPSK	1920	1/2	2.67×	No
QPSK_2560bps	QPSK	2560	1/2	2×	No
QPSK_5120bps	QPSK	5120	1/2	1×	Yes
QPSK_10240bps	QPSK	10240	None	None	Conditional

Table 2: PayloadMode Summary

Mode	Summary
FH_80bps	Extreme low-SNR
BPSK_640bps	Most robust coherent
BPSK_960bps	Long-range, low-SNR
BPSK_1280bps	Medium/long range
BPSK_2560bps	Baseline BPSK
QPSK_1280bps	Backward compatible
QPSK_1920bps	Backward compatible
QPSK_2560bps	Backward compatible
QPSK_5120bps	High-rate coded mode
QPSK_10240bps	Requires external FEC

A.5 Measured SNR Waterfall Thresholds

Table 3 summarizes the approximate SNR breakpoints observed in representative coastal environments where packet error rate increases sharply.

Table 3: 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

A.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

Index