# iAcommsDriver

## Brief Overview

iAcommsDriver is an interface for the WHOI uModem to allow both complete access to modem statistics and facilitate easy transmission and reception of data using the modem. Toby Schneider’s Goby libraries are used for communication with the modem. Most statistics and received data are published to more than one MOOS variable for easier use by other applications and thorough logging.

## Configuration Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Type** | **Default** | **Description** |
| PortName | String |  | Serial port name |
| ID | Integer |  | ID used by this node for filling in source in uModem packets. |
| PSK\_minipackets | Boolean | False | Use fsk or psk encoding for mini packets. |
| enable\_ranging | Boolean | False | Enable synchronization to PPS for one-way ranging. |
| show\_range\_pulses | Boolean | True | Publish range pulses for visualization in pMarineViewer. |
| in\_sim | Boolean | false | Configure for simulation use |
| Community (global) | String |  | Used to set vehicle name. |

## Subscriptions

Shaded cells are deprecated -prefer use of ACOMMS\_TRANSMIT.

|  |  |  |
| --- | --- | --- |
| **Variable** | **Type** | **Description** |
| ACOMMS\_TRANSMIT | Binary string | Unified data transmission message provided by HoverAcomms library |
| ACOMMS\_TRANSMIT\_DATA | String | Passing ascii data to driver for transmission |
| ACOMMS\_TRANSMIT\_DATA\_BINARY | Binary string | Passing binary data to driver for transmission |
| ACOMMS\_TRANSMIT\_RATE | Double | Integer rate |
| ACOMMS\_TRANSMIT\_DEST | Double | Integer ID of destination (0 for broadcast) |
| NAV\_X, NAV\_Y | Double | Used for posting of range pulses |
| LOGGER\_DIRECTORY | String | to put log file into same directory as pLogger |
| ACOMMS\_TRANSMITTED\_REMOTE | Binary string | Simulation only – record of transmissions by all instances of iAcommsDriver |

## Publications

Shaded cells are deprecated – prefer use of ACOMMS\_RECEIVED and ACOMMS\_TRANSMITTED.

|  |  |  |
| --- | --- | --- |
| **Variable** | **Type** | **Description** |
| ACOMMS\_RECEIVED\_DATA | Binary string | Data received in a transmission |
| ACOMMS\_RECEIVED | Binary string | Unified data received message provided by HoverAComms library |
| ACOMMS\_TRANSMITTED | Binary string | Unified data transmitted message provided by HoverAcomms library |
| ACOMMS\_RECEIVED\_DATA\_HEX | String | Received data in hex format |
| ACOMMS\_BAD\_FRAMES | String | Comma delimited list of bad frames |
| ACOMMS\_TRANSMITTED\_DATA\_HEX | String | Transmitted data in hex format |
| ACOMMS\_RECEIVED\_ALL | String | DebugString of received ModemTransmission protobuf |
| ACOMMS\_RECEIVED\_SIMPLE | String | Brief summary of reception |
| ACOMMS\_TRANSMIT\_SIMPLE | String | Brief summary of transmission |
| ACOMMS\_DRIVER\_STATUS | String | status of driver, updated every 5 seconds |
| ACOMMS\_DRIVER\_WARNING | String | For debugging information |
| VIEW\_RANGE\_PULSE | String | Posting of range pulses on transmission or reception |
| ACOMMS\_IMPULSE\_RESPONSE | String | Raw CAIRE message from modem |
| ACOMMS\_SNR\_OUT, ACOMMS\_SNR\_IN, ACOMMS\_DQR | Double | Data picked from ACOMMS\_RECEIVED\_ALL for ease of access by other applications |

Proposed changes to reduce overall number of variables and simplify. In general, applications should use the ACOMMS\_RECEIVED\_ALL posting (or ACOMMS\_RECEIVED if that doesn’t work out (see italics below)) to get all they’re data. The HoverAcomms library provides all the necessary parsing functionality. Transmissions will remain as is and support both the integrated and simple posting formats. The new acomms parser will parse all the ACOMMS\_RECEIVED\_ALL and ACOMMS\_TRANSMIT\_ALL messages in a log file into a matlab/excel friendly format.

Receiving variables:

* ACOMMS\_RECEIVED\_ALL – same as current functionality, should be used as primary source of data by other programs and parsed using library functions – *need to check if we can both log this and give binary data to other applications with one message, if not may need a binary version as well*
* ACOMMS\_RECEIVED\_HEX – hex copy of data for logging purposes
* ACOMMS\_IMPULSE\_RESPONSE

Transmitting variables

* ACOMMS\_TRANSMIT\_ALL – generated in the same way as ACOMMS\_RECEIVED\_ALL
* ACOMMS\_TRANSMIT\_HEX – hex copy of transmitted data for logging purposes

Ranging variables:

* ACOMMS\_REMUS\_LBL – comma delimited list of reported ranges
* ACOMMS\_RANGING\_TIME – for when synced to pps, or receiving a two-way ranging reply

Legacy variables (can be toggled)

* ACOMMS\_RECEIVED\_DATA
* ACOMMS\_BAD\_FRAMES

Misc. other postings

* VIEW\_RANGE\_PULSE – pulses for pmarineviewer
* ACOMMS\_DRIVER\_WARNING – various warnings
* ACOMMS\_DRIVER\_STATUS – posting of driver status every 5 seconds or on change
* SYSTEM\_TIME\_SECONDS – posting of current system time every 5 seconds

When running in simulation, several other posting may also be used to exchange data behind the scenes.

## Basic Usage

Use of the new transmitting and receiving methods is preferred, but both old and new should work. All publications, both new and old format, will be made regardless of the method used to initiate a transmission.

### Driver status

The driver will publish its current status to ACOMMS\_DRIVER\_STATUS at least once every 5 seconds. Status can be “transmitting”, “receiving”, “ready”, or “not running” (only occurs at startup). Transmission requests will be ignored if the driver is not ready.

### Transmitting – new version

Use the AcommsTransmission class provided by the HoverAcomms library to construct a complete transmission request included rate, destination, and data. The serialized version of this message should be posted to ACOMMS\_TRANSMIT in binary format. If the driver is ready the message will be sent immediately and the details of the transmission will be published using the same AcommsTransmission class to ACOMMS\_TRANSMITTED (binary) and ACOMMS\_RECEIVED\_ALL (non-binary). If the driver is not ready then the transmission request will be ignored and a warning issued.

### Transmitting – Deprecated version

The transmission rate is set using the ACOMMS\_TRANSMIT\_RATE variable. See the uModem documentation for a complete listing of possible rates and the size of their data payloads. 13-bit mini-packets can be transmitted by setting rate 100. See section 1.7 for more details on sending mini-packets. Transmit destination is set using the ACOMMS\_TRANSMIT\_DEST variable. For now, only use the default value of 0 (broadcast).

Transmission is initiated when data is posted to either ACOMMS\_TRANSMIT\_DATA or ACOMMS\_TRANSMIT\_DATA\_BINARY. You must use the binary variable if your data contains the byte 0x00. Data will automatically be packaged into frames according to the set rate and truncated if necessary. The driver will post a hex translation of the transmitted data (post truncation) to ACOMMS\_TRANSMITTED\_DATA\_HEX and a brief summary of the transmission information will be posted to ACOMMS\_TRANSMIT\_SIMPLE. A yellow range pulse is posted emanating from the transmitter’s location if range pulses are enabled.

### Receiving – new version

All information for each reception is published as a single message to ACOMMS\_RECEIVED using the AcommsReception class defined in the HoverAcomms library. Use of this class and its included member functions should simplify applications that used to subscribe to multiple variables for acomms reception information.

### Receiving - deprecated version

All receptions should be accompanied by a posting to ACOMMS\_RECEIVED\_ALL containing all receive information, including statistics. A brief summary will be posted to ACOMMS\_RECEIVED\_SIMPLE. If data was received, it will be posted to ACOMMS\_RECEIVED\_DATA as binary and ACOMMS\_RECEIVED\_DATA\_HEX as a hex translation. Multiple frames will be concatenated together before publication. A comma delimited string of the indices of bad frames is published to ACOMMS\_BAD\_FRAMES, but no placeholder is included with the received data publication.

As a simple example we explore a hypothetical transmission (note this is not an actual micromodem transmission type). Consider a packet consisting of 4 frames sized 2 bytes each. On the transmitter we post to ACOMMS\_TRANSMIT\_DATA the string “abcdefghi”. The string is truncated and split into frames to be transmitted: “ab”, “cd”, “ef”, and “gh”. The middle two frames are lost. On the receiver the string “abgh” is published to ACOMMS\_RECEIVED\_DATA and “2,3” is published to ACOMMS\_BAD\_FRAMES to indicate that the 2nd and 3rd frames were lost.

If there are no bad frames an empty string will be published to ACOMMS\_BAD\_FRAMES. A posting of “-1” indicates that no frames were received.

The raw impulse response message from the modem is caught and posted to ACOMMS\_IMPULSE\_RESPONSE, primarily for logging purposes. Individual statistics can be posted as their own variables for ease of use. Currently snr\_in, snr\_out, and dqr are posted individually.

## Message Formats

The new message formats used for the variables ACOMMS\_TRANSMIT, ACOMMS\_RECEIVED, and ACOMMS\_TRANSMITTED are all defined in the HoverAcomms library.

ACOMMS\_RECEIVED\_ALL is created by calling the DebugString() method on the ModemTransmission protobuf. Line endings are replaced with the placeholder “<|>”. The simple acomms parser source code can be used as reference for decoding this and other goby protobuf structures.

ACOMMS\_TRANSMIT\_SIMPLE and ACOMMS\_RECEIVED\_SIMPLE are defined in lib\_acomms\_messages.

Hex formatted messages use colon delimiters between bytes. For example the phrase “Hello world” would be posted as “48:65:6c:6c:6f:20:77:6f:72:6c:64”. Hex values less than 10 will be posted using one digit instead of two (e.g. “61:0:61”).

## Minipackets (rate 100)

Minipackets can carry 13 bits of information passed in two bytes. The micromodem will always perform a bitwise and with 0x1f on the first byte. If only a single byte is passed to the driver for transmission, it will be packed with 0x00 in the first position. See the following examples:

acomms\_transmit\_data\_binary --> acomms\_received\_data  
a) 0x6161 --> 0x0161  
b) 0x0061 --> 0x0061  
c) 0x6100 --> 0x0100  
d) 0x61 --> 0x0061

ACOMMS\_TRANSMITTED\_DATA\_HEX can be used to check the data actually being transmitted in a minipacket.

## Logging

The driver writes a separate “goby log” in the same folder used by pLogger for the MOOS logs. This log includes all of the raw nmea sentences exchanged between the goby uModem driver and the uModem hardware. File names are goby\_logX.txt where X is an integer that is incremented as needed if the driver is restarted. Because iAcommsDriver depends on a publication for pLogger to determine the logging directory, it cannot be run before pLogger is started.

## Typical bridging setup

When using pAcommMonitor, which is usually run on the shoreside computer, ACOMMS\_RECEIVED and ACOMMS\_TRANSMITTED should be bridged from all vehicles to the shoreside.

If running the acomms driver in simulation mode then ACOMMS\_TRANSMITTED should also be bridge from the shoreside to all vehicles under the alias ACOMMS\_TRANSMITTED\_REMOTE.

# Lib\_acomms\_messages

Library used for passing acomms related messages containing multiple pieces of information.

## SIMPLIFIED\_RECEIVE\_INFO

### Fields

|  |  |  |
| --- | --- | --- |
| **field** | **type** | **description** |
| Vehicle name | String | Name of the vehicle that sent the transmission |
| Source | Integer | Source id of the transmitter |
| Rate | Integer | Transmission rate (100 for mini) |
| Num frames | Integer | Total number of expected frames |
| Num good frames | Integer | Number of frames correctly received |
| Num bad frames | Integer | Number of frames with errors |

### Format

“vehicle\_name,%s:source,%d:rate,%d:num\_frames,%d:num\_good\_frames,%d:num\_bad\_frames,%d”

## SIMPLIFIED\_TRANSMIT\_INFO

### Fields

|  |  |  |
| --- | --- | --- |
| **field** | **type** | **description** |
| Vehicle name | String | Name of the vehicle that sent the transmission |
| Rate | Integer | Transmission rate (100 for mini) |
| Dest | integer | Destination ID (0 for broadcast) |
| Num frames | Integer | Total number of frames sent |

### Format

“vehicle\_name,%s:rate,%d:dest,%d:num\_frames,%d”

# uPokeDBHex

## Brief Overview

uPokeDBHex is essentially the same as uPokeDB, except that it works for binary strings instead of normal strings. It cannot be used to poke normal strings, but it will still display their contents albeit in hex notation.

## Usage

Exactly the same as uPokeDB for doubles. When poking binary strings, use hex notation with colons to separate bytes.

uPokeDBHex ACOMMS\_TRANSMIT\_DATA\_BINARY=”68:65:6c:6c:6f”

The value of string and binary string variables will also be displayed in hex format after being poked.

# iHoverKayak

## Brief Overview

Interface with the low level control running on the Arduino.

## Configuration

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Type** | **Default** | **Description** |
| PORT\_NAME | String | /dev/ttyO0 | Serial port name (not required since all use the hardware /dev/ttyO0) |
| BAUD\_RATE | Integer | 115200 | Default 115200 – no need to change |
| INVERT\_RUDDER | Boolean | False |  |
| RUDDER\_OFFSET | Double | 0 |  |
| RADIO\_WAIT\_TIME | Integer | 120 | time in seconds to wait for confirmation when switching radio power before switching back |

## Subscriptions

|  |  |  |
| --- | --- | --- |
| **Variable** | **Type** | **Description** |
| DESIRED\_THRUST | double | Thrust output (-100 to 100) |
| DESIRED\_RUDDER | double | Rudder angle (-90 to 90, but further limited by Arduino) |
| RADIO\_POWER | String | “freewave” or something else |

## Publications

|  |  |  |
| --- | --- | --- |
| **Variable** | **Type** | **Description** |
| VOLTAGE | double | Battery voltage as measured by motor driver |
| CPU\_BOX\_TEMP | Double | Temperature in degrees Celsius of cpu box |
| ROBOTEQ\_HEATSINK\_TEMP | double |  |
| ROBOTEQ\_INTERNAL\_TEMP | double |  |
| ROBOTEQ\_BATTERY\_CURRENT | double |  |
| ROBOTEQ\_MOTOR\_CURRENT | double |  |
| CPU\_BOX\_CURRENT | double |  |
| ARDUINO\_THRUST | double | Actual output thrust by motor driver |
| ARDUINO\_RUDDER | double | Currently just the set point |

## Radio Power Switching

Radio power is switched immediately when posting to RADIO\_POWER. If the same value is not posted again to RADIO\_POWER within RADIO\_WAIT\_TIME seconds then the driver will switch back.