Communication Protocol Datasheet

The communication between the odroid and Arduino works over UART. It follows a packet based system. The protocol does not incorporate ACK/NAK, error detection, or data requests. Both parties will send information as they see fit. The system does use heartbeats to verify the other party is alive and in a state to receive commands.

Packets

START – 0xAA (1 byte)	Message ID (1 byte)	Payload(1 – 30 bytes)

The packets are extremely simple and ARE susceptible to framing errors. Meaning packets may get lost or corrupted if the START byte exists in the message ID or Payload and the system isn't aligned. This should occur very rarely because once the transmitter and receiver are aligned, it is very unlikely they will become unaligned. Additionally, most of the packets send will be streamed at a high rate. So if one packets doesn't make it, it doesn't really matter because the system will realign by the time the next packet is sent.

Messages

Message definitions are predefined. This protocol must be implemented by both parties for proper decoding. Some messages are designed to be sent by only the Arduino and others only by the odroid. Some can be sent by both.

Sent by Arduino:

CHANNELS_IN

Description	Decoded PPM signals from RC receiver; Meant to be streamed at a constant rate(~40hz) at all times				
Message ID	0x01				
Message Length	10 bytes				
Field	**Type**	**Unit**	**Min**	**Max**	**Description**

Throttle	uint16_t	microseconds	900	2100	Throttle channel
Steering	uint16_t	microseconds	900	2100	Steering channel
AUX1	uint16_t	microseconds	900	2100	Extra channel
AUX2	uint16_t	microseconds	900	2100	Extra channel

Sent by Odroid:

CONTROL

Description	Set actuator positions using computer control; Meant to be streamed at a constant rate(~30hz) during computer control				
Message ID	0x02				
Message Length	10 bytes				
Field	**Type**	**Unit**	**Min**	**Max**	**Description**
Throttle	uint16_t	microseconds	900	2100	Throttle channel
Steering	uint16_t	microseconds	900	2100	Steering channel
AUX1	uint16_t	microseconds	900	2100	Extra channel
AUX2	uint16_t	microseconds	900	2100	Extra channel

SET_MODE

Description	Attempt to change mode on Arduino; Sent to request a mode change				
Message ID	0x03				
Message Length	3 bytes				
Field	**Type**	**Unit**	**Min**	**Max**	**Description**

Mode	uint8_t	MODE_ENUM	0	255	Mode

Bidirectional:

HEARTBEAT

Description	Sent to notify party other of presence and contains additional system state; Streamed at all times at a constant rate(~2hz) by both parties				
Message ID	0x00				
Message Length	4 bytes				
Field	**Type**	**Unit**	**Min**	**Max**	**Description**
Mode	uint8_t	MODE_ENUM	0	255	Current Arduino mode. This field can be ignored when sent from odroid
Errors	uint8_t	BITMASK	0	255	Error bitmask

DEBUG

Description	Message for sending debug values; It can be sent by both parties, but it was designed to debug on the Arduino				
Message ID	0x04				
Message Length	14 bytes				
Field	**Type**	**Unit**	**Min**	**Max**	**Description**
Value 0	float	n/a			Debug value
Value 1	float	n/a			Debug value
Value 2	float	n/a			Debug value

Modes

Modes are a data type used by SET_MODE and HEARTBEAT messages to signify or change the mode on the Arduino.

Mode	**ID**	**Description**
MANUAL	0x00	The driver is in control
AUTO	0x01	The computer is in control
FAILSAFE	0x02	The system encountered a critical error and is in standby.
INVALID	0xFF	Not sure why I made this. Implemented by has no use

Error bitmask

The heartbeat message has an error bitmask to indicate Arduino system status. This bitmask is currently not defined. This should probably be called status bitmask but oh well. Additionally, this field could be used to express errors in terms of error IDs instead of a bitmask

Bit Number	Name	Description
0	N/A	0 = ; 1 = ;
1	N/A	
2	N/A	
3	N/A	
4	N/A	
5	N/A	
6	N/A	
7	N/A	