# ME218C 2022 Communications Protocol

Version 2.1.8, May 18th, 2022

**Communications Committee Members:**
Joaquin Castillo
Carissa Cirelli
Josh DeWitt
Nicholas Riley
Andrew Sack
Melody Yang

a.k.a. AJ and the Anonymous Animals

## Revision Log

| Revision Log | | |
|---|---|---|
| Date | Section/Description | Authors |
| 5/3/22 | Initial Draft | Joaquin, Carissa, Josh, Nick, Andrew, Melody |
| 5/9/22 | Added clarifications around fuel level | Andrew |
| 5/9/22 | Changed Yaw byte to relative, clarified symmetric input, control mixing | Josh |
| 5/9/22 | Updated pairing state diagrams to include valid control/status received events | Joaquin |
| 5/10/22 | Made CW positive in accordance with axis image | Andrew |
| 5/13/22 | Draft of example “Entire Packets” | Josh |
| 5/14/22 | Changed API Identifier for Transmission FROM XBee to 0x81 | Andrew |
| 5/17/22 | Switched COMM_TIMEOUT from 5s to 3s<br>Elaborated on meaning of COMM_TIMEOUT | Joaquin, Carissa, Josh, Nick, Andrew, Melody |
| 5/18/22 | Updated Team 1 XBee addresses | Joaquin |

## Introduction

A long time ago in a galaxy far, far away… Six intrepid teams gathered to do battle on the hallowed grounds of Terman Fountain. One guild will emerge victorious, but our destinies are still unknown…

Update 5/25/22: One guild is better than the other.

# Overview of XBee Communication

## XBee Data Frame Structure

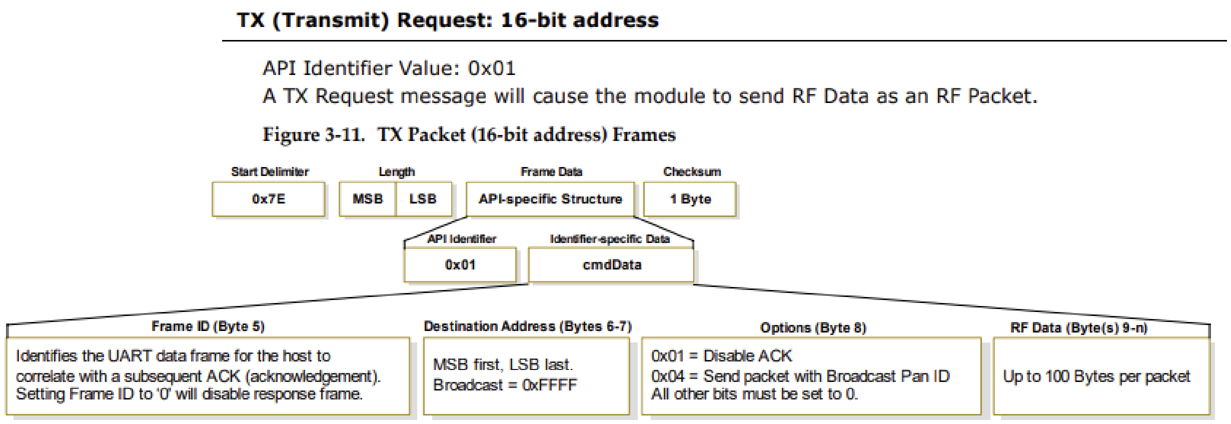
Transmit (PIC to XBee)

| Description | Start Delimiter | Length | | API Identifier | Frame ID | Destination Address | Options | RF Data | Checksum |
|---|---|---|---|---|---|---|---|---|---|
| Byte # | 1 | 2 | 3 | 4 | 5 | 6-7 | 8 | 9-N | n+1 |
| Value | 0x7E | MSB | LSB | | | | | US | |

Receive (XBee to PIC)

| Description | Start Delimiter | Length | | API Identifier | Destination Address | RSSI | Options | RF Data | Checksum |
|---|---|---|---|---|---|---|---|---|---|
| Byte # | 1 | 2 | 3 | 4 | 5-6 | 7 | 8 | 9-N | n+1 |
| Value | 0x7E | MSB | LSB | | | | | US | |

## XBee API Commands Used



**TX (Transmit) Request: 16-bit address**

API Identifier Value: 0x01

A TX Request message will cause the module to send RF Data as an RF Packet.

**Figure 3-11. TX Packet (16-bit address) Frames**

Identifies the UART data frame for the host to correlate with a subsequent ACK (acknowledgement). Setting Frame ID to '0' will disable response frame.

MSB first, LSB last.
Broadcast = 0xFFFF

0x01 = Disable ACK
0x04 = Send packet with Broadcast Pan ID
All other bits must be set to 0.

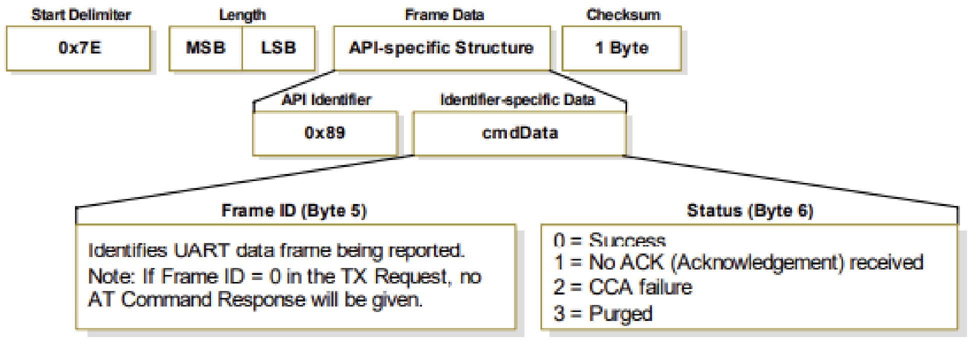
Up to 100 Bytes per packet

## TX (Transmit) Status

API Identifier Value: 0x89

When a TX Request is completed, the module sends a TX Status message. This message will indicate if the packet was transmitted successfully or if there was a failure.

**Figure 3-12. TX Status Frames**



NOTES:

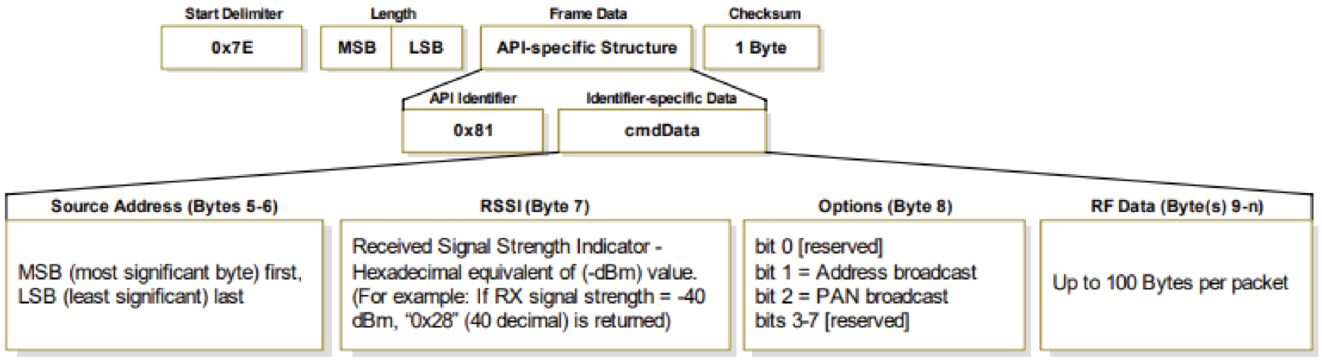
- "STATUS = 1" occurs when all retries are expired and no ACK is received.
- If transmitter broadcasts (destination address = 0x000000000000FFFF), only "STATUS = 0 or 2" will be returned.
- "STATUS = 3" occurs when Coordinator times out of an indirect transmission. Timeout is defined as (2.5 x SP (Cyclic Sleep Period) parameter value).

## RX (Receive) Packet: 16-bit address

API Identifier Value: 0x81

When the module receives an RF packet, it is sent out the UART using this message type.

**Figure 3-14. RX Packet (16-bit address) Frames**



Start Delimiter
0x7E
Length
MSB
LSB
Frame Data
API-specific Structure
Checksum
1 Byte
API Identifier
0x81
Identifier-specific Data
cmdData
Source Address (Bytes 5-6)
MSB (most significant byte) first, LSB (least significant) last
RSSI (Byte 7)
Received Signal Strength Indicator - Hexadecimal equivalent of (-dBm) value. (For example: If RX signal strength = -40 dBm, "0x28" (40 decimal) is returned)
Options (Byte 8)
bit 0 [reserved]
bit 1 = Address broadcast
bit 2 = PAN broadcast
bits 3-7 [reserved]
RF Data (Byte(s) 9-n)
Up to 100 Bytes per packet

### Team XBee Addresses

| Team | TUG XBee address | PILOT XBee address |
|---|---|---|
| 0 | 0x2187 | 0x218A |
| 1 | 0x2086 | 0x2084 |
| 2 | 0x2184 | 0x2186 |
| 3 | 0x2188 | 0x2183 |
| 4 | 0x2085 | 0x2082 |
| 5 | 0x2185 | 0x2087 |

## Summary of Commands

- Pairing
- Control - Thrust information along with Option Byte and refuel sent at 5 Hz
- Status (fuel) - The boat must respond to all control messages with the fuel level

### List of Message IDs for Commands

| Header | Command |
|---|---|
| 0x01 | Control (PILOT->TUG) |
| 0x02 | Status (TUG->PILOT) |
| 0x03 | Request to Pair (PILOT->TUG) |
| 0x04 | Pairing Acknowledged (TUG->PILOT) |

## Pairing Protocol

Both the TUG and PILOT have a pairing functionality. When the device is in the PAIRED state, the device will not transmit any pairing commands and will ignore any pairing commands they receive as well as any commands from a device other than the one it is paired to. When the device is in ATTEMPTING TO PAIR state, it will ignore any non-pairing commands it receives. The TUG/PILOT shall enter ATTEMPTING TO PAIR mode when:

1. Powered On
2. The PAIRING BUTTON is pressed

3. No message is received for COMM_TIMEOUT (COMM_TIMEOUT = 3s)

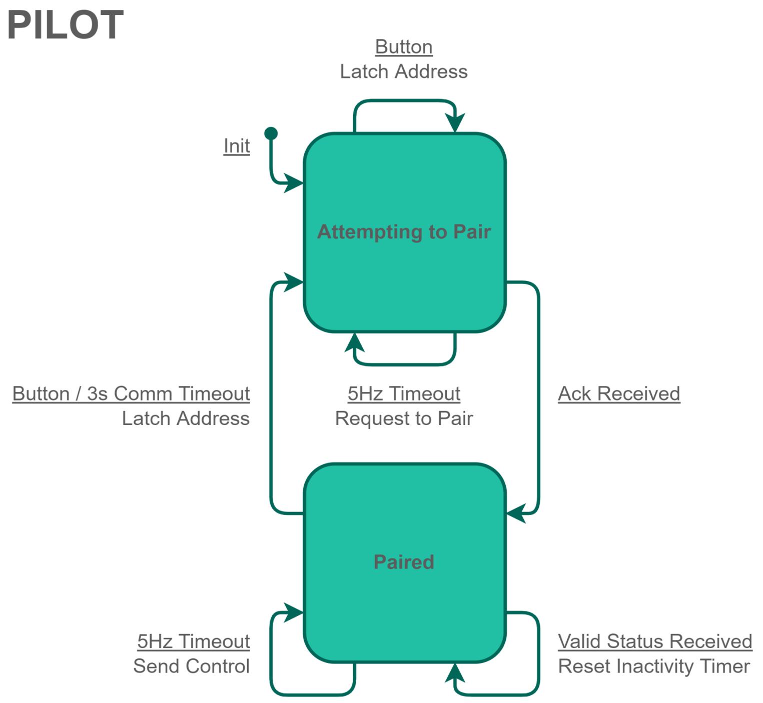
When the PILOT enters ATTEMPTING TO PAIR mode it will transmit a REQUEST TO PAIR message to the TUG address specified by the CONCON upon entering ATTEMPTING TO PAIR mode. It will repeat this transmission at 5 Hz until a PAIRING ACKNOWLEDGED message is received from the specified TUG.

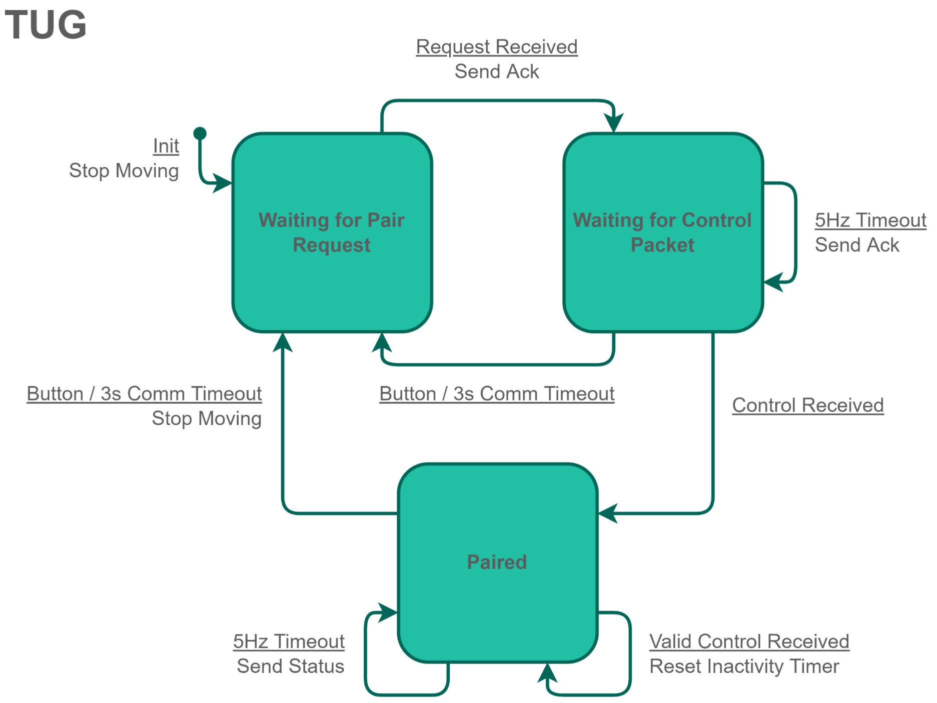
The TUG address selector should only be read when the PAIRING BUTTON is pressed or the PILOT is powered on. On COMM TIMEOUT, the address selector should not be read, and the last latched address should be used. This should attempt to reconnect the PILOT with the same TUG in case of a communications error during a round.

When the TUG enters WAITING FOR PAIR REQUEST mode it will wait until it receives a REQUEST TO PAIR message from a PILOT. It will store the PILOT's address and reply with a PAIRING ACKNOWLEDGED message. It will repeat this transmission at 5 Hz until a CONTROL message is received from the specified CONCON. When the TUG enters ATTEMPTING TO PAIR it shall also reset the Fuel Level to full (255).

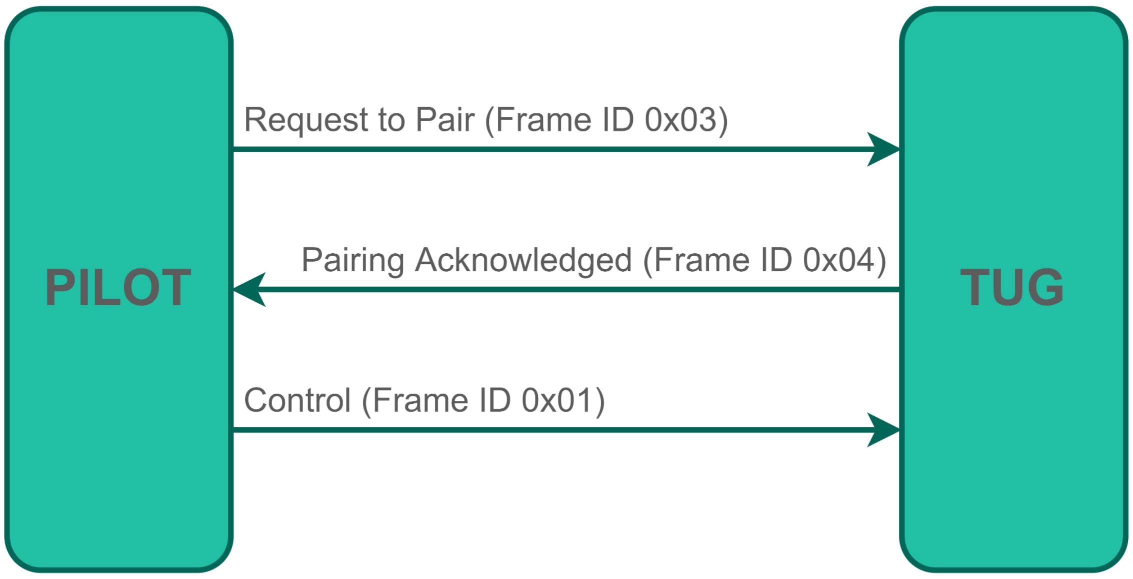
COMM_TIMEOUT kicks both PILOT and TUG back to the initial pairing stage (ATTEMPTING TO PAIR and WAITING FOR PAIR REQUEST, respectively) when they have not received a valid message for that period of time. In the absence of a change in address selector, this should repair the same PILOT - TUG combination and allow the game to continue.

# Pairing State Diagrams





## Pairing Packet Flow


PILOT
TUG
Request to Pair (Frame ID 0x03)
Pairing Acknowledged (Frame ID 0x04)
Control (Frame ID 0x01)

## Pairing Frame Data

### PILOT -> TUG Request to Pair

| Byte # | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Meaning | Message ID | TUG Addr MSB | TUG Addr LSB | PILOT Addr MSB | PILOT Addr LSB | ACK |
| Value | 0x03 | uint8_t | uint8_t | uint8_t | uint8_t | 0xAA |

### TUG -> PILOT Pairing Acknowledgement

| Byte # | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Meaning | Message ID | TUG Addr MSB | TUG Addr LSB | PILOT Addr MSB | PILOT Addr LSB | ACK |
| Value | 0x04 | uint8_t | uint8_t | uint8_t | uint8_t | 0x55 |

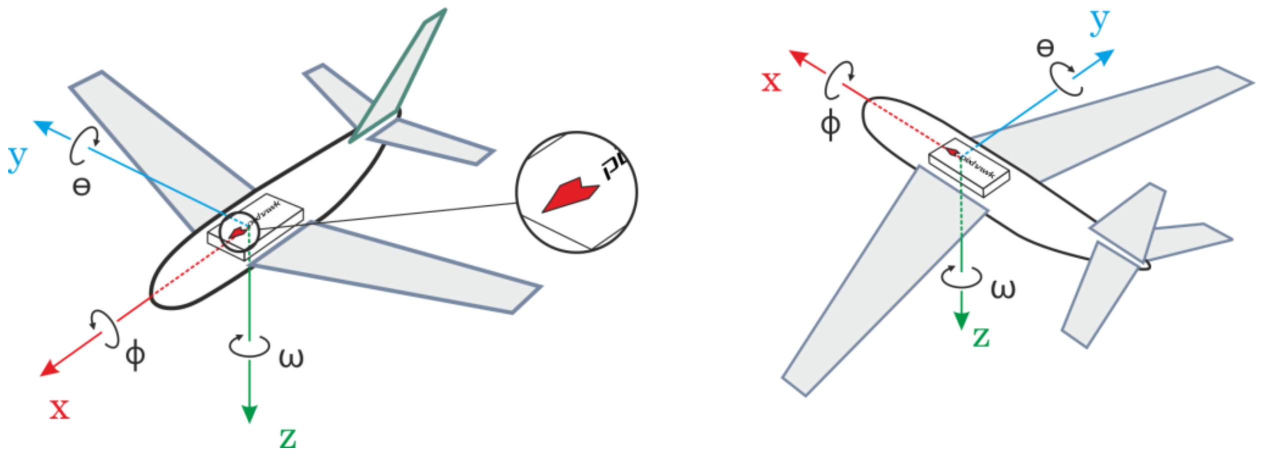
# Control Protocol

## Control Frame Data

### PILOT -> TUG: Control Frame Data

| Byte # | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Meaning | Message ID | X | Y | Yaw | Refuel | Mode3 |
| Value | 0x01 | int8_t | int8_t | int8_t | uint8_t | uint8_t |

#### Definition of Axes



https://docs.px4.io/v1.12/en/config/flight_controller_orientation.html

#### Control Mixing

The 2 dimensional control scheme is a modified Arcade drive. The X and Yaw values can map to a standard tank drive, per the standard Arcade method, while the Y value adds a translational component. See the following sections for the detailed implementation of each byte.

The exact mixing of each drive component is left to the team to decide, but should conform to the existing fuel consumption guidelines, and not involve any data not explicitly declared inside the X, Y, and Yaw control data.

While each byte must provide the functions defined below, each team is allowed to invert or mix their signals however they like on the PILOT side, i.e. moving a Yaw joystick to the right creates a CCW rotation.

See the link below for a brief overview of Arcade style driving, as well as an interactive demo at the bottom (most helpful):
https://xiaoxiae.github.io/Robotics-Simplified-Website/drivetrain-control/arcade-drive/

### Symmetric Control Values

To make all motion mostly symmetric, while the full range of a signed 8 bit integer is -128 to +127, each control integer value should only range from -127 to +127. In the case of a -128 being read by the controller, a value equivalent to -127 should be assigned and acted up.

### Conversion between Signed and Unsigned Integers

Because the XBee protocol relies on UART, data must be converted into unsigned 8 bit numbers ("uint8_t"). A recommended way to do this is to calculate the control value as an int8_t, cast the value as a uint8_t when inserting into the TX buffer on the PILOT side, and then cast as an int8_t when reading from the RX buffer into the TUG control register

### Control.X (Fore and Aft system, forward to backward)

This byte controls the forward/backward motion of the TUG. 0 is no motion, +127 is full throttle forward, and -127 is full throttle backward.

### Control.Y (Port to Starboard, left to right)

This byte controls the left to right, or Port and Starboard motion of the TUG. Per the axes above, positive Y is to the Right (Starboard). 0 is no side to side motion, +127 is full throttle to starboard, and -127 is full throttle to port.

### Control.Yaw

This byte controls the rate of rotation of the TUG about the Z axis. Per the axes above, -127 is max angular speed CounterClockWise turn, +127 is max angular speed ClockWise turn.

### Control.Refuel

The refuel byte indicates when refueling is completed. 0 means no refuel action has occurred, and 1 means refueling has occurred. The TUG should only respond to the refueling action when its fuel level is 0. The PILOT should continue sending 1 until it receives full fuel from the TUG, upon which it should set the refuel byte to 0 again. The PILOT may not reset the refuel byte back to 1 until it receives 0 fuel from the TUG.

### Control.Mode3

This is an "auxiliary" byte, to be used to actuate "fun" and aesthetic devices on the TUG. The ability to send or read to this byte is not required for any TUG or PILOT, and if not implemented, should be ignored/send all zeros.

Each bit represents one togglable “action”, so that up to 8 different actions can be triggered. If multiple actions are implemented, their associated bits should start at the Least Significant Bit and move up to MSB for action 8.

## Status Protocol

### Status Frame Data

#### TUG -> PILOT: Status Frame Data

| Byte # | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Meaning | Message ID | Fuel Level | UNUSED | UNUSED | UNUSED | UNUSED |
| Value | 0x02 | uint8_t | 0x00 | 0x00 | 0x00 | 0x00 |

##### Status.FuelLevel

The fuel level byte indicates fuel level. 255 is full and 0 means you need to refuel! It should decrease by 0.255 per % thrust per second, rounded to the nearest integer for transmission.

Integration should be calculated at a rate of at least 5 Hz, and controlled by a timer that is independent of Control/Status packets. The value of remaining fuel should be tracked by at least a single precision floating point value. At each integration step, the total output of the motors should be summed, multiplied by a timestep, and then subtracted from the remaining fuel level.

## Refueling Procedure

### TUG

TUG’s Fuel Level:
0:
If the TUG is at 0 fuel, it shall ignore all motor controls and disable all TUG actuators. It shall remain in this state until it receives 1 for the Control Message Refuel bit. When it receives a 1, the TUG’s fuel level shall be set to 255.
non-0:
The TUG shall ignore all Refuel bit values.

## PILOT

If the PILOT receives a fuel level that is not 0 from the TUG, it shall set the GASCON's refuel flag to 0.

## Acknowledgements

The ME218C 2022 communications committee would like to ACK the valuable support of Professor Ed Carryer and Karl Gumerlock in the creation of this standards document.

# Appendix

## Sample Messages

Color Legend:

| | | |
|---|---|---|
| XBee Protocol | API Protocol | ME218C 2022 Protocol |

For the following examples, the PILOT has an address of 0x2142 and the TUG has an address of 0x2169. The messages TO the XBee are sent, and the same messages are received in the FROM table.

The Pairing message is a sample pairing acknowledgement, sent from TUG to PILOT.
The Control message is a control frame sent from PILOT to TUG.
The Status message is sent from TUG to PILOT.

An important difference between the sent and received message is the location of the Source/Destination address

These message frames are provided as reference, please review the sections above for implementation details.

## Entire Message Frame - Transmission TO XBee

| Byte Index | Pairing | Control | Status | Sample Pairing ACK Frame | Sample Control Frame | Sample Status Frame |
|---|---|---|---|---|---|---|
| 1 | Start Delimiter | | | 0x7E | 0x7E | 0x7E |
| 2 | Length MSB | | | 0x00 | 0x00 | 0x00 |
| 3 | Length LSB | | | 0x0B | 0x0B | 0x0B |
| 4 | API Identifier | | | 0x01 | 0x01 | 0x01 |
| 5 | Frame ID | | | 0x01 | 0x01 | 0x01 |
| 6 | Destination Address MSB | | | 0x21 | 0x21 | 0x21 |
| 7 | Destination Address LSB | | | 0x42 | 0x69 | 0x42 |
| 8 | Options | | | 0x00 | 0x00 | 0x00 |
| 9 | Message ID | | | 0x04 | 0x01 | 0x02 |
| 10 | TUG Addr MSB | X | Fuel Level | 0x21 | +127 = 0x7F | 136 = 0x88 |
| 11 | TUG Addr LSB | Y | - | 0x69 | 0 = 0x00 | 0x00 |
| 12 | PILOT Addr MSB | Yaw | - | 0x21 | -64 = 0xC0 | 0x00 |
| 13 | PILOT Addr LSB | Refuel | - | 0x42 | 0x00 | 0x00 |
| 14 | ACK | Mode 3 | - | 0x55 | 0x01 | 0x00 |
| 15 | Checksum | | | (0xFF-0xAB) = 0x54 | (0xFF-0xCD) = 0x32 | (0xFF-0xED) = 0x22 |

## Entire Message Frame - Transmission FROM XBee

| Byte Index | Pairing | Control | Status | Sample Pairing ACK Frame | Sample Control Frame | Sample Status Frame |
|---|---|---|---|---|---|---|
| 1 | Start Delimiter | | | 0x7E | 0x7E | 0x7E |
| 2 | Length MSB | | | 0x00 | 0x00 | 0x00 |
| 3 | Length LSB | | | 0x0B | 0x0B | 0x0B |
| 4 | API Identifier | | | 0x81 | 0x81 | 0x81 |
| 5 | Source Address MSB | | | 0x21 | 0x21 | 0x21 |
| 6 | Source Address LSB | | | 0x69 | 0x42 | 0x69 |
| 7 | RSSI | | | 0xA4 = -164dB | 0x10 = -16dB | 0x45 = -69dB |
| 8 | Options | | | 0x00 | 0x00 | 0x00 |
| 9 | Message ID | | | 0x04 | 0x01 | 0x02 |
| 10 | TUG Addr MSB | X | Fuel Level | 0x21 | +127 = 0x7F | 136 = 0x88 |
| 11 | TUG Addr LSB | Y | - | 0x69 | 0 = 0x00 | 0x00 |
| 12 | PILOT Addr MSB | Yaw | - | 0x21 | -64 = 0xC0 | 0x00 |
| 13 | PILOT Addr LSB | Refuel | - | 0x42 | 0x00 | 0x00 |
| 14 | ACK | Mode 3 | - | 0x55 | 0x01 | 0x00 |
| 15 | Checksum | | | (0xFF-0x75) = 0x8A | (0xFF-0xB5) = 0x4A | (0xFF-0x5A) = 0xA5 |