Communications Protocol

ME 218C Spring 2016

Table of Contents:

Section 1: Definitions

Term	Definition
LOBBYIST	A remote-controlled hovercraft capable of moving forward/backward, left/right and optionally performing a digitally controlled special action.
PAC	A controller capable of interfacing with any of the LOBBYISTs. Required inputs are: pairing action, fwd/back, left/right and digital special action.
pairing action	A command input that, when performed, will initiate a pairing sequence between the PAC and the LOBBYIST.

Section 2: Communications Overview

This document describes the protocol to be used for communication between remote controlled hovercraft (LOBBYIST) and its controller (PAC). On the day of the project presentation, a tournament will take place. In each round (lasting 218 seconds), two teams of three will compete against each other (red team and blue team). Each team will get three random PACs, and there will be a total of four LOBBYISTs on the field. Each LOBBYIST must be able to interface with any of the PACs. A successful pairing will allow a PAC to control the LOBBYIST for 45 seconds or until the connection gets interrupted. The task of the LOBBYISTs is to herd lawmakers (ping pong balls) into revolving doors (goals) to score points. The arena will have two reversers, plates that will temporarily convert their respective goals into the goal of opposite color.

The communication between PACs and LOBBYISTs will be carried out using XBee radio modules working in API mode. The maximum frequency of exchanges between a PAC and its LOBBYIST will be 5Hz. This will need to be enforced by the PAC using its timers to disable requests for 20 ms after each transmitted packet. In order to take control of a LOBBYIST, the PAC will send a pairing request to it. The LOBBYIST will then respond with a status packet acknowledging the pairing. After a successful pairing, a 32 byte key will be sent to the LOBBYIST by the PAC. From then on, the PAC will send control packets to the LOBBYIST, with rolling encryption. Every command must be acknowledged by the LOBBYIST through the use of a status packet. Following a 1 second lack of transmissions, or after 45 seconds after the initial pairing, the PAC and the LOBBYIST will unpair from each other and go into the free state. A paired LOBBYIST will ignore any incoming pairing requests.

Section 3: Hardware Requirements

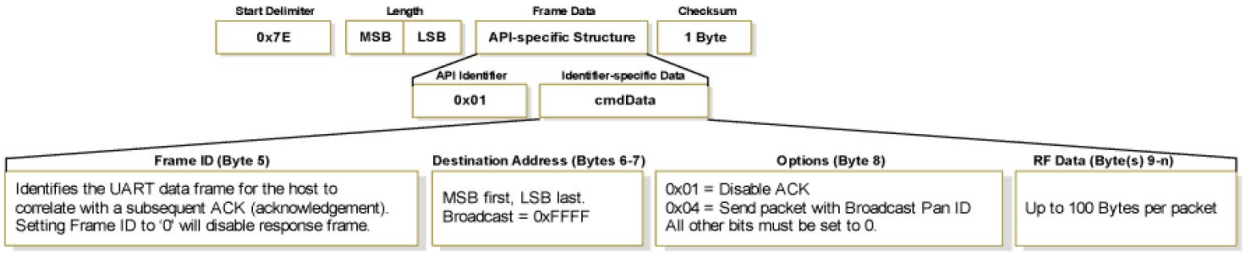
In order to implement the radio communication described in this document, the following minimum hardware requirements are necessary:

- All teams will use XBee radio modules (Zigbee) to implement radio communication between PAC and LOBBYIST.
- Both the LOBBYIST and PAC will carry one of these modules on board. Each radio module will have an address associated with it (2180 through 218B) and (2080 through 208B). Each robot will also have an ID number assigned using a hardware attachment.
- The communication between any micro-controller and the XBee radio module should take place via asynchronous serial communication only. (UART)
- The asynchronous serial communication between any onboard micro-controller and any Xbee board will take place at a fixed bit rate of 9600.
- The PAC and LOBBYIST can transmit at a maximum rate of 5Hz each. That includes failed transmissions. It also means that a response can occur faster than 200ms after a message is received.

Section 4: Packet Structure

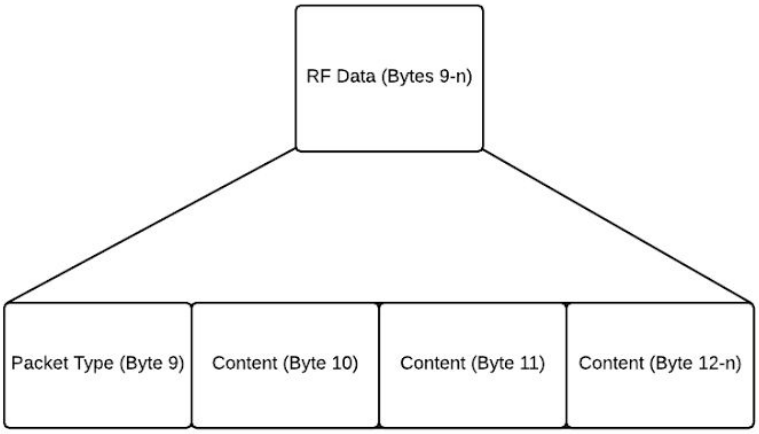
4.1 XBee API Packet Structure:

The figure below provides reference for the structure of the XBee TX packet. All the packets described throughout section 4 are contained within the RF Data section of the transmission.



Note: the Frame ID byte has no impact on the communications between the PAC and the LOBBYIST. It is only used to identify which outgoing packet an XBee status message refers to. It's required that the LOBBYIST receives a confirmation of receipt from the PAC's XBee before rotating the its encryption key. For that, the Frame ID has to be used but this document does not impose any implementation details.

4.2 RF Data Structure:



4.3 Packet Types:

Packet type	**Direction**	**Header**	**Encrypted**	**Contents [First Byte → Last Byte]**
REQ_PAIR	P → L	0x00	N	[PAIR_DATA]
ENCR_KEY	P → L	0x01	N	[ENC_KEY0][ENC_KEY1] ..... [ENC_KEY31]
CTRL	P → L	0x02	Y	[CTRL0][CTRL1][CTRL2][CHKSM]
STATUS	L → P	0x03	N	[PAIR] [ACKDATA]

† Letter P refers to PAC and L to Lobbyist

4.4 Pair Request Packet:

- Broadcast (destined for specific hovercraft using the ID number)
- Packet length: Header + 1 data byte

Purpose:
The pair request packet is broadcast from a PAC seeking to pair with an unpaired Lobbyist. The Lobbyist address is stored in bits 0 through 6 of the PAIR_DATA byte and the color is stored in bit 7.

Byte 0: [HEADER]
REQ_PAIR= 0x00

Byte 1: [PAIR_DATA]
The PAIR_DATA byte is used by the PAC to indicate which Lobbyist number (LBY) it wants to pair with.

PAIR_DATA

COLOR	NUM6	NUM5	NUM4	NUM3	NUM2	NUM1	NUM0

- Bits<6-0>: The address number of a Lobbyist for playing in the game.
  - While grading or game, only bits 1 and 0 will be used as up to four lobbyists will be present. Bits 2 to 6 should be cleared. The transmitted number should represent the LOBBYIST number minus one.
  - For debugging, the user should program their LOBBYIST's number to be equal to their class team number + 3.
- Bit 7: COLOR The color selector.
  - If set, assigns the LOBBYIST to the blue team. Otherwise the LOBBYIST will be red.

4.5 Encryption Key Packet:

- Directed, unencrypted, PAC to paired Lobbyist
- Packet length: Header + 32 data bytes

Purpose:
The 32 byte encryption key, starting with ENC_KEY0. Each byte is a random number generated by the PAC every time pairing is performed.

Byte 0: [HEADER]
ENCR_KEY= 0x01

Byte 1: [ENC_KEY]
This is the 32 byte encryption key, sent from the zero index byte up to the 31st byte. This is a random number generated by the PAC. Once paired, all of the control packet data sent from the PAC to the Lobbyist (including the header byte) are encrypted using this key, until the end of the period of control.

ENC_KEY0-ENC_KEY31- 32 byte encryption key

KEYBIT7	KEYBIT6	KEYBIT5	KEYBIT4	KEYBIT3	KEYBIT2	KEYBIT1	KEYBIT0

N.B.: Send ENC_KEY0 first.

NOTE: For testing, it may be helpful to use an encoder key of all FF's. This way, the encrypted data will just be complemented.

4.6 Control Packet:

- Directed, encrypted
- Packet length: Header + 4 data bytes

Purpose:
The CTRL bytes are sent from the PAC to the LOBBYIST and are used to control the motion of the LOBBYIST, as well as any special actions the LOBBYIST might take. Note that all the bytes in the control packet (including the packet header) are encrypted with the rolling key.

Byte 0: [HEADER]
CTRL= 0x02

Byte 1: [CTRL0]
All 8 bits of the CTRL0 byte are used to create one signed integer, which specifies the forward and backward speed of the LOBBYIST. When the CTRL0 byte is a positive value, the LOBBYIST will move forward. Conversely, when the CTRL0 byte is a negative value, the LOBBYIST will move backward. When the CTRL0 byte has a value of 0, the LOBBYIST's propulsion should be off. This byte can store a signed integer ranging in value from -128, which represents reversal at full speed, to +127, which represents full speed forward.

CTRL0 - Analog Forward/Backward Control

FB7	FB6	FB5	FB4	FB3	FB2	FB1	FB0

- Bits <7:0> - Signed 8 bit integer

Byte 2: [CTRL1]
All 8 bits of the CTRL1 byte are used to create one signed integer, which specifies the directional control for the LOBBYIST. When the CTRL1 byte is a positive value, the LOBBYIST will turn to the right. Conversely, when the CTRL1 byte is a negative value, the LOBBYIST will turn left. When the CTRL1 byte has a value of 0, the LOBBYIST should not turn in either direction. This byte can store a signed integer ranging in value from -128, which represents turning left at the fastest speed possible, to +127, which represents turning right at the fastest speed possible.

CTRL1 - Analog Left/Right Control

LR7	LR6	LR5	LR4	LR3	LR2	LR1	LR0

- Bits <7:0> - Signed 8 bit integer

Byte 3: [CTRL2]
The CTRL2 byte is used to specify any special actions a LOBBYIST can take. These actions include braking, purposely unpairing with the PAC, and any other optional special actions designated at the discretion of each team*.
*Note: Teams who use bits 2-7 should be aware that other PACs may not be able to control these special actions because they will not have user input modes available to control them.

CTRL2 - Special Actions

SA5	SA4	SA3	SA2	SA1	SA0	UNPR	BRK

- Bits <7:2> - Special actions (7 to 3 optional, SA0 must be available in every PAC)*
  - 1 = Activate special action
  - 0 = Deactivate special action
- Bit 1 – Unpairing
  - 1 = Unpair
  - 0 = Remain Paired
- Bit 0 - Brake
  - 1 = Apply brake
  - 0 = Disengage brake

*Unimplemented special action bits should be cleared.

Byte 4: [CHKSM]
CHKSM is used as an error checking measure to ensure the decryption process remains in sync. Every time a control packet is sent by the PAC, control bytes 0, 1, 2, and 3 should be summed together and the result should be placed in CHKSM (i.e. byte 4 of the control packet data). Once the control packet has been received by the LOBBYIST, the packet should be decrypted and the LOBBYIST should s+um bytes 0, 1, 2, and 3. If the result of this summation does not match the value sent in byte 4, an error occurred and the decryption process should be treated as being out of sync. If this is the case, the status packet should be updated such that the Pairing bit in STATUS is cleared and the Decryption Failure bit in STATUS is set.

CHKSM - Decryption Error Checking

CHKSM7	CHKSM6	CHKSM5	CHKSM4	CHKSM3	CHKSM2	CHKSM1	CHKSM0

- Bits <7:0> - Unsigned 8 bit integer

4.7 Status Packet:

- Directed, semi encrypted (data byte 1 unencrypted, data byte 2 encrypted)

- Packet length: Header + 2 data bytes

Purpose:
The STATUS packet is sent from the LOBBYIST to the PAC and is used to let the PAC know it is connected to the LOBBYIST. The receipt of the packet is also a signal that the sent command has been acknowledged. If the packet sent to the LOBBYIST was not encrypted (either a Pair Request or an Encryption Key), then the DEC_ERR bit is clear. The STATUS packet also contains ACKDATA, the CTRL Data check sum byte (encrypted) previously received, to tell the PAC which instruction is being acknowledged.

Byte 0: [HEADER]
STATUS= 0x03

Byte 1: [PAIR]
Bit 0 in the PAIR byte is used to show whether the LOBBYIST is paired to a PAC or not.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	DEC_ERR	PAIR

- Bit <7:2>: Unimplemented - read/write as zero
- Bit 1: Decryption error bit
  - 1 = Decryption failure occurred
  - 0 = Decryption successful or not applicable
- Bit 0: Pairing bit
  - 1 = PAC and LOBBYIST are paired
  - 0 = PAC and LOBBYIST are unpaired

Byte 2: [ACKDATA]
The ACKDATA byte being sent back to the PAC contains the encrypted CHKSM byte of the CTRL data previously sent to the lobbyist that has been acknowledged by it. If the PAC fails to receive the STATUS data bytes from the lobbyist within 100 ms, it shall resend the previous control byte sequence to the lobbyist. However, if the acknowledge is received before the 100 ms timeout, the PAC can check the ACKDATA to confirm the receive of its past CTRL data and can now prepare to send new CTRL data.

ACKDATA 7	ACKDATA 6	ACKDATA 5	ACKDATA 4	ACKDATA 3	ACKDATA 2	ACKDATA 1	ACKDATA 0

- Bits <7:0> - Unsigned 8 bit integer

Section 5: Encryption Protocol

Purpose:
To prevent de-synchronization between the PAC and the LOBBYIST

How to:

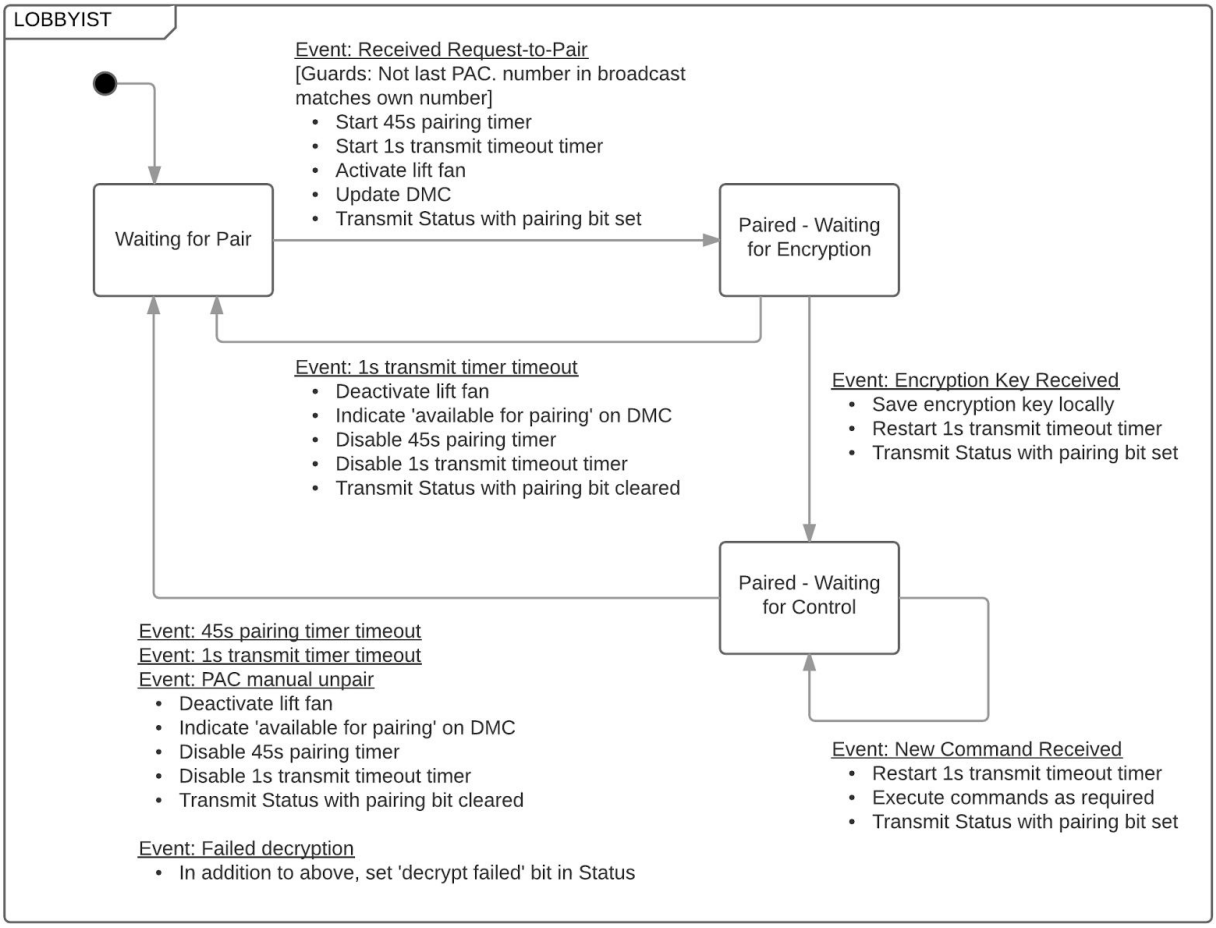
The encryption protocol consists of an XOR cipher with a 32 byte key sequence randomly generated by the PAC. The XOR cipher is implemented by performing a bitwise XOR operation on the transmitted data packet, using the current encryption byte as the operand. The key should be generated every time a new pairing process is initiated between a lobbyist and the PAC. Post pairing, every byte of the data packet sent by the PAC to the lobbyist will be encrypted using the current byte of the key. Note the data packet consists only of the RF DATA part of the XBee packet. That includes all the bytes of the control packet defined in section 4. The key will continue rotating between transmissions with the index incrementing with each byte of data transmitted.

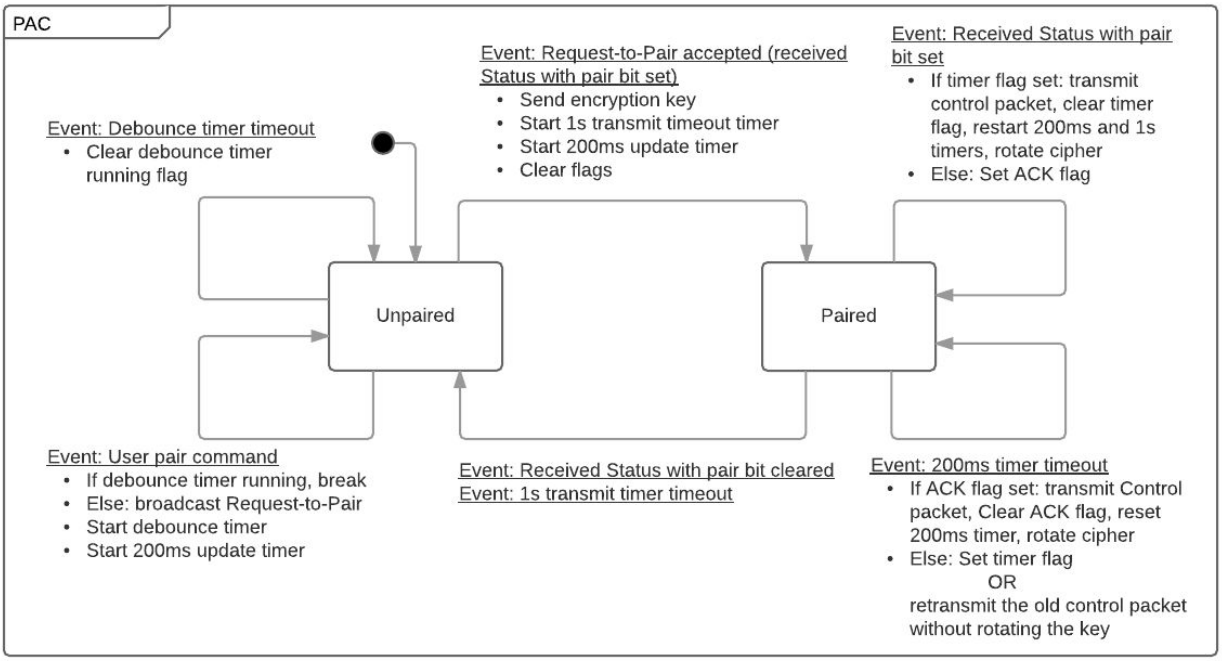
To ensure that the PAC is always in synchronization with the LOBBYIST, an additional checksum byte will be required to be transmitted at the end of each control packet. The checksum byte is defined as the sum of the header and content bytes in the control packet before encryption. The checksum byte will be treated like a normal data byte i.e it will be encrypted with the continuing key index before being transmitted as part of the data frame.

At the Lobbyist, the additional checksum byte (last byte) of the control packet is decrypted and compared with the sum of all the other decrypted bytes of the control packet to establish the legitimacy of the data received.

If the decrypted checksum matched the sum of the decrypted bytes of the control packet, the packet received is confirmed as legitimate and the pair bit continues to be set in the STATUS byte that is sent back. If the decrypted checksum does not match the sum, the decryption error bit is set in the STATUS. In addition, the Lobbyist will clear the pair bit in the STATUS byte and unpair with the PAC. In this case, the PAC and the LOBBYIST will have to re-establish pairing with a fresh key sequence to resume communication.

Section 6: Communication State Charts





Notes to the PAC state diagram:
-if no ACK is received after sending a command, the PAC can resend the same message after 200ms and keep trying until the 1s timer expires. However, until it gets an ACK, it cannot rotate the key.

Section 7: Sample Communications Sequences

Interaction	PAC		LOBBYIST
Pairing attempt (LOBBYIST unavailable to pair)	Transmits **Request to Pair** (broadcast, unencrypted) with number of targeted LOBBYIST	⇒	If already paired, ignores all broadcast messages and messages directed towards other LOBBYISTs
	Ready to transmit Request to Pair after 200ms		
Pairing attempt (LOBBYIST available to pair)	Transmits **Request to Pair** (broadcast, unencrypted) with number of targeted LOBBYIST and team color	⇒	If LOBBYIST number in Request to Pair matches own number: ● Indicates successful pairing on DMC ● Indicates team color specified by Request to Pair on DMC ● Starts 45s pairing timer ● Starts 1s transmit timeout timer ● Activates lift fan ● Ignores any further broadcast

			messages
		⇐	Transmits **Status** (unencrypted, directed) with pairing bit set
	Generates and transmits **Encryption Key** (unencrypted, directed) to paired LOBBYIST	⇒	If received transmission is directed to this radio, saves key locally ● Resets 1s transmit timeout timer
		⇐	Transmits **Status** (unencrypted, directed) with pairing bit set
	Ready to start transmitting Control packets		Ready to start receiving Control packets
Control sequence (maximum 5 Hz)[1]	Encrypts **Control** packet with accordance with section 5		
	Transmits **Control** packet (encrypted, directed) to paired LOBBYIST	⇒	If received transmission is directed to this radio: ● Decrypts Control packet with locally saved encryption key ● Validates Control packet contents by running and comparing Control packet checksum
			If decrypted checksum *valid*: ● Resets 1s transmit timeout timer ● Implements control commands ● Increments encryption key index in accordance with section 5
	Increments encryption key index in accordance with section 5	⇐	Transmits **Status** (unencrypted, directed) with pairing bit set
	Ready to start transmitting next Control packet		Ready to start receiving next Control packet
Control sequence (decrypt error)	Encrypts **Control** packet with 32-byte encryption key in accordance with section 5		
	Sends **Control** packet (encrypted, directed) to paired LOBBYIST	⇒	If received transmission is directed to this radio: ● Decrypts Control packet with locally

[1] The Control packet transmit-receive sequence depends on the PAC radio receiving a Status packet from the LOBBYIST radio to acknowledge receipt of each Control packet. Depending on the delay required for the PAC to receive acknowledgement from the LOBBYIST, the effective rate may be slower than 5 Hz.

			saved encryption key in accordance with section 5 ● Validates Control packet contents by running and comparing Control packet checksum
			If decrypted checksum *invalid*: ● Deactivates lift fan ● Indicates available-for-pairing on DMC ● Disables 45s pairing timer ● Disables 1s transmit timeout timer
		⇐	Transmits **Status** (unencrypted, directed) with pairing bit *cleared* and decrypt error bit set
	Ready to transmit Request to Pair		Ready to receive Request to Pair
Communication link lost (transmit timeout timer expired)			If 1s transmit timeout timer expires: ● Deactivates lift fan ● Indicates available-for-pairing on DMC ● Disables 45s DMC pairing timer ● Disables 1s transmit timeout timer
		⇐	Transmits **Status** (unencrypted, directed) with pairing bit *cleared*
	Ready to transmit Request to Pair		Ready to receive Request to Pair
Pairing timer timeout			If 45s pairing timer expires: ● Deactivates lift fan ● Indicates available-for-pairing on DMC ● Disables 45s pairing timer ● Disables 1s transmit timeout timer
		⇐	Transmits **Status** (unencrypted, directed) with pairing bit *cleared*
	Ready to transmit Request to Pair		Ready to receive Request to Pair
Manual PAC unpair	Sends **Control** packet (encrypted, directed) to paired LOBBYIST with unpair bit set	⇒	
			● Deactivates lift fan ● Indicates available-for-pairing on DMC ● Disables 45s pairing timer ● Disables 1s transmit timeout timer

		⇐	Transmits **Status** (unencrypted, directed) with pairing bit *cleared*
	Ready to transmit Request to Pair		Ready to receive Request to Pair