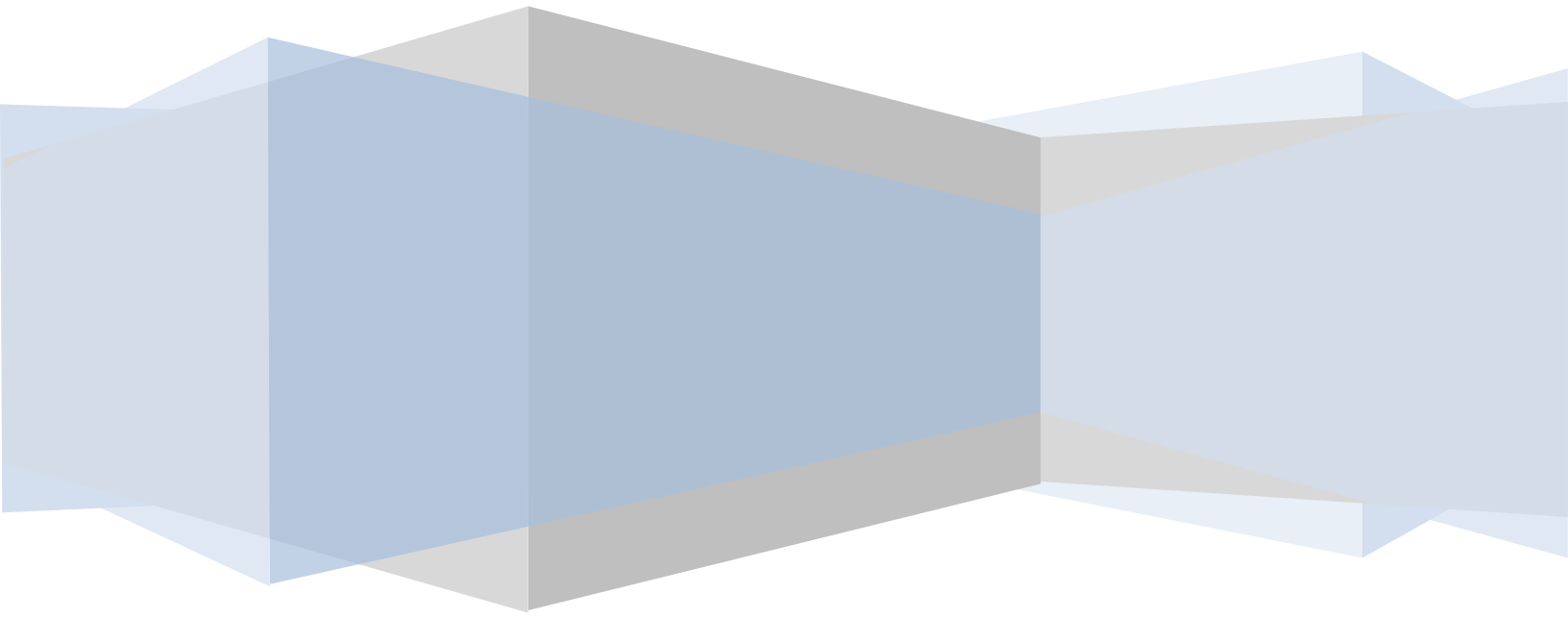
218c 2011-2012

ME 218c Communications Protocol

The Communications Committee



Communications Overview

The communication protocol specified by the document requires each user-data section of the XBee packet have an identifier byte, Byte 0, corresponding to the type of data.

Pairing/Handshaking

At the start of every game, the LUCs and HZVs must establish communication via pairing/handshaking. This process allows for LUCs to take control of specific HZVs. This data structure is two bytes long. Byte 0 is the data-type identifier and byte 1 is the pairing/handshaking data. Communication will be initiated when the LUC sends out a pairing/handshaking message with the second byte as 0xED. When the HZV receives this message successfully, it will return the message data structure with the second byte as 0xCD. If the LUC is able to receive this message from the HZV, successful pairing/handshaking is established. The LUC can then take control of the HZV through gameplay commands described below.

Necromancer Commands

The Necromancer commands will allow modification of zombie behavior. HZV's that are in human mode should ignore the commands received from the Necromancer. These commands will enable the modification of the maximum speed of the zombie (ranging from 50% to 75%), the turning rate (50% to 100%) and confuse operator (swap left and right). The length of the data of the Necromancer command will be 5 bytes. Byte 0 will be the data-type identifier byte followed by 4 bytes corresponding to one of the actions specified above.

***The data structure for the Necromancer will be specified by the teaching staff. Bytes 1 through 4 are reserved for the data, as is specified.*

Gameplay Commands

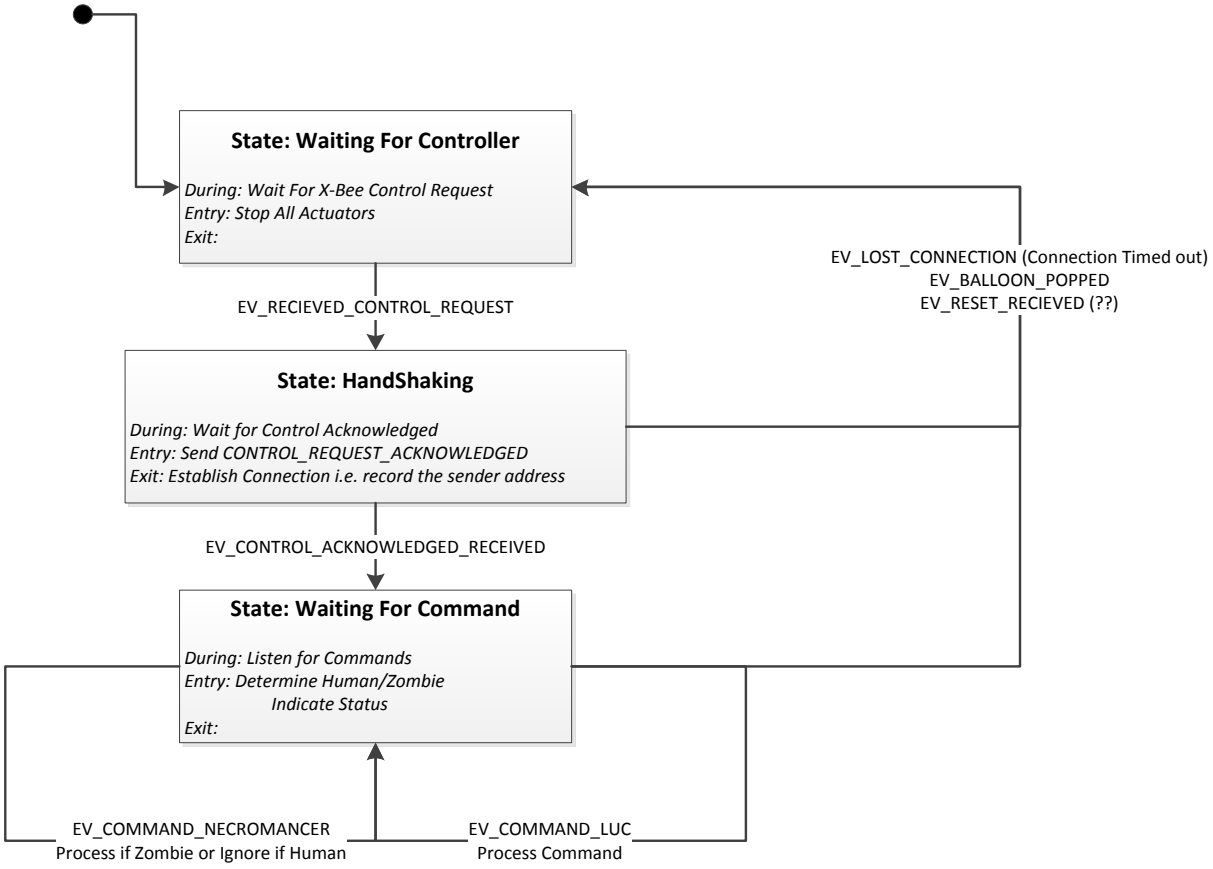
The gameplay commands correspond to the navigation/action data structure. It is specified that this data structure of 7 bytes long, where Byte 0 is the data-type identifier. Bytes 1-6 are command bytes corresponding to the actions implemented on the HZV. Byte 1 corresponds to speed of the HZV, which is followed by a byte for direction and a byte for attack action. The following 3 bytes are specified for custom functionality, which will vary from boat to boat.

Communication Failure

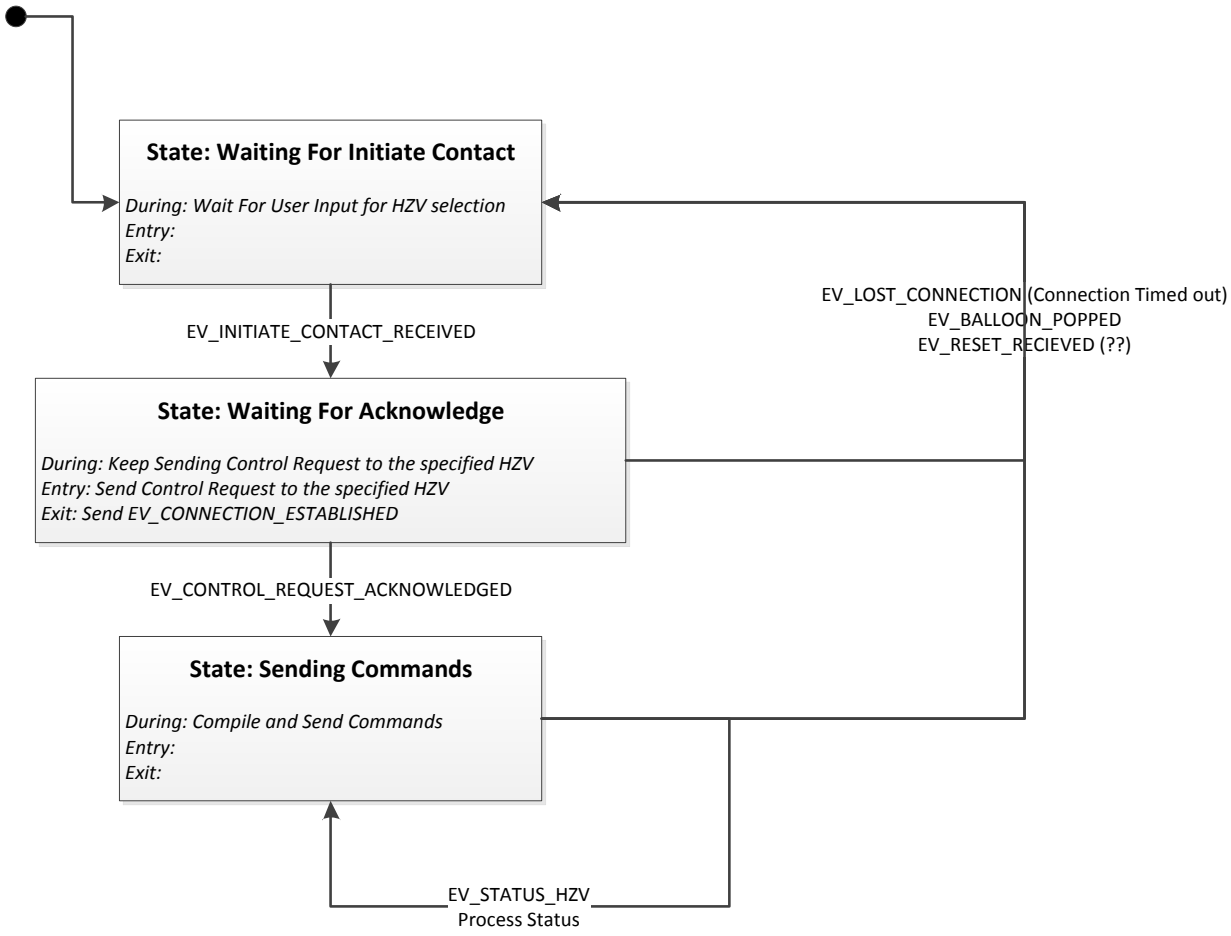
Communication failure is occurs when the HZV hasn't received a new command from the LUC for 1 second. As new commands are being sent at the rate of 5Hz, this would mean that there have been 5 consecutive dropped commands. When communication failure occurs, the HZV is required to go into **Waiting_For_Controller** state and indicate that communication failure has occurred. This will allow LUCs on deck to make an attempt to gain control of the HZV.

State chart

Human/Zombie Vehicle (HZV) state chart



Living/Undead Controller (LUC) state chart



Zigbee Protocol in ME218c Communication Applications

Both HZV and LUC microcontrollers will interface with the local XBee device over via UART communication. Several kinds of messages will be passed back and forth over UART:

1. The microcontroller will send frames containing data for the XBee to send wirelessly
2. The XBee will send the microcontroller frames containing data it has received wirelessly
3. The XBee will send the microcontroller status messages in response to send requests

The data structures based on the ZigBee API are detailed in the following sections.

Zigbee API: Send Data Frame (Microcontroller to XBee)

The Zigbee API data for all frames is structured as follows:

Type – API data	**Length**
1.	Start Delimiter 1 Byte (0x7E)
2.	Length of Data 2 Bytes (n)
3.	Frame Data n Bytes
4.	Checksum 1 Byte

The Frame Data byte is sub-divided into two main parts:

Type – Frame Data	**Length**
1.	API Identifier 1 Byte **(0x01 for sending data)**
2.	Command Data n-1 Bytes

The Command Data section is further sub-divided into 4 main parts, the last of which contains the user data. The Frame ID can be independently set for each packet of data. However, for the 218c application, all communication data is resent in every packet and therefore, the response to missed frames would be the same as communicating the next frame. For this reason, the Frame ID will not be governed by our protocol, but will be available for use if teams would like to use it for troubleshooting. **It is recommended that teams do not use 0x00 for their Frame ID**, as this option disables acknowledgement messages from the XBee back to the microcontroller.

Type – Command Data	**Length**
1.	Frame ID 1 Byte (team-determined, not 0x00)
2.	Destination Address 2 Bytes
3.	Options 1 Byte
4.	**User Data to send** **n-5 Bytes**

When sending a packet of data, the Frame ID will need to be followed by the Destination Address of the desired recipient. Upon receiving a data packet, the bytes following the Frame ID will indicate the sender of that packet. The user data structure is the ME 218c communication protocol, whose structure is described in the following section. The length of the user data byte will vary based on the type of data being sent.

The checksum byte is the modulo-127 sum of all the frame data bytes.

Zigbee API: Result From a Send Data Frame (XBee to Microcontroller)

Upon receiving a data packet wirelessly from another XBee, the XBee will pass this frame to the local microcontroller. The XBee API data structure is constant:

Type – API data	**Length**
1.	Start Delimiter 1 Byte (0x7E)
2.	Length of Data 2 Bytes (n)
3.	Frame Data n Bytes
4.	Checksum 1 Byte

The Frame Data byte is again sub-divided into two main parts:

Type – Frame Data	**Length**
1.	API Identifier 1 Byte **(0x89 for transmission response)**
2.	Command Data 2 Bytes

The Command Data section is sub-divided as follows:

Type – Command Data	**Length**
1.	Frame ID Confirm 1 Byte **(repeats team-determined Frame ID)**
2.	Status 1 Byte **(0, 1, 2, or 3)**

Per the XBee API, the Status byte will be 0x00 for a successful transmission, 0x01 if no ACK has been received from the destination radio 0x02 if the channel is not clear.

Zigbee API: Received Data Frame (XBee to Microcontroller)

Upon successful receipt of a send data frame, the XBee will reply to the microcontroller with this data frame. The XBee API data structure is constant:

Type – API data	**Length**
1.	Start Delimiter 1 Byte (0x7E)
2.	Length of Data 2 Bytes (n)
3.	Frame Data n Bytes
4.	Checksum 1 Byte

The Frame Data byte is again sub-divided into two parts:

Type – Frame Data	**Length**
1.	API Identifier 1 Byte **(0x81 for received data)**
2.	Command Data 2 Bytes

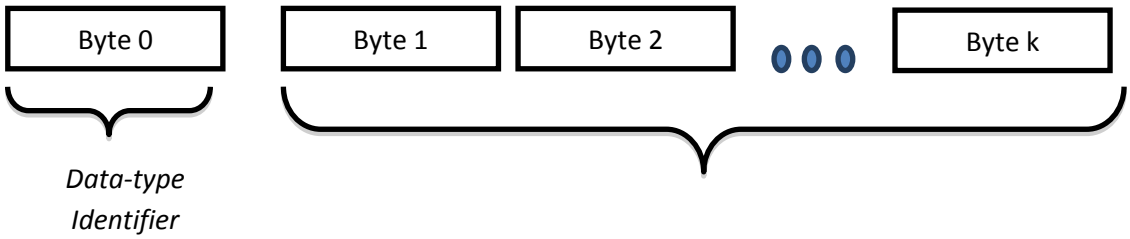
The Command Data section is sub-divided as follows:

Type – Command Data	**Length**
1.	Frame ID 1 Byte **(determined by sending device)**
2.	Source Address 2 Bytes **(address of sending device)**
3.	Options 1 Byte
4.	**User Data received** **n-5 Bytes**

The user data structure is the ME 218c communication protocol, whose structure is described in the following section. The length of the user data byte will vary based on the type of data received.

ME 218c Communication Data

ME218c-Proposed 'User Data' structure



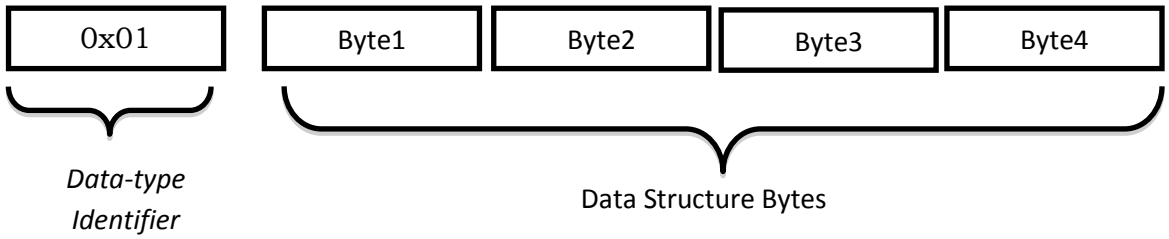
The proposed data structure identifies the first byte as a Data-type identifier, which is used to extract information regarding the incoming data. The following bytes correspond to the data as structured by the communications committee.

Data-type Identifier – Byte 0

This byte of transmission identifies the type of data being sent. The Hex value identifies the type of data. The length of each corresponding data type is given.

Identifier		Data Type	Length
0x01	------	Necromancer data	5 Bytes
0x02	------	Navigation/Action data (From LUC to HZV)	7 Bytes
0x03	------	Handshaking Procedure data	2 Bytes
0x04	------	HZV Acknowledgement	1 Byte

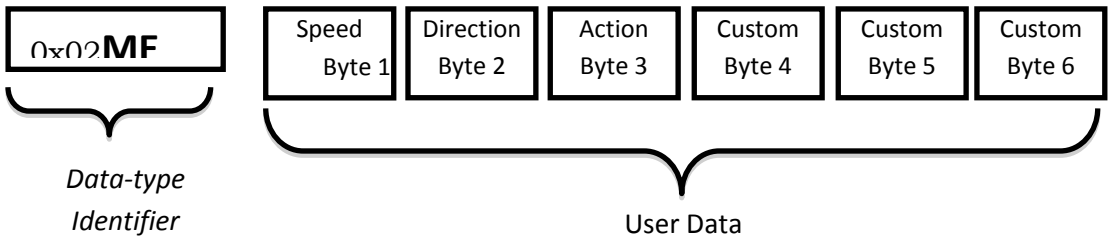
Necromancer data byte (0x01)



***The data structure for the Necromancer will be specified by the teaching staff. Bytes 1 through 4 are reserved for the data, as its specified.*

Navigation/Action Data structure (0x02)

The navigation data structure consists of the data-type identifier (0x02) followed by a speed byte, a direction byte, an action byte and makes allowance for 3 custom bytes to be sent. It requires that the first 2 custom bytes be of analog format and the last custom byte is digital format.



The ***Speed Byte*** and the ***Direction Byte*** consist of a signed char. Based on this we have 255 bit resolution to specify speed ranging from *100% forward* to *100% reverse* and 255 bit resolution to specify direction ranging from *100% right* to *100% left.*

Speed Logic Table

127 ---- Full Speed Forward (100%)

0 ---- Stop (0%)

-128 ---- Full Speed Reverse (-100%)

Speed Byte							
R/W – 1	*R/W – 0*	*R/W – 0*	*R/W – 0*	*R/W – 0*	*R/W – 0*	*R/W – 0*	*R/W – 0*
SPD7	**SPD6**	**SPD5**	**SPD4**	**SPD3**	**SPD2**	**SPD1**	**SPD0**
Bit 7							*Bit 0*

Direction Logic Table

127 ---- Turn RIGHT (100%)

0 ---- Go Straight (0%)

-128 ---- Turn LEFT (100%)

Direction Byte							
R/W – 1	*R/W – 0*	*R/W – 0*	*R/W – 0*	*R/W – 0*	*R/W – 0*	*R/W – 0*	*R/W – 0*
DIR7	**DIR6**	**DIR5**	**DIR4**	**DIR3**	**DIR2**	**DIR1**	**DIR0**
Bit 7							*Bit 0*

The ***Action Byte*** is reserved for the "Attack" operation of the balloon popping mechanism. The setting of 'Bit 0' of Byte 3 structure triggers the attack action. As long as Bit 0 stays hi, the attack device will stay deployed. The HZV is responsible for retracting the attack mechanism after 2 seconds of deployment if Bit 1 is still set. It should ignore the fact that the button is still depressed if the 2 seconds has expired. Redeploying the attach mechanism should necessitate clearing and resetting Bit 0.

Bit 0 HI ---- Deploy Attack Mechanism (for up to 2 seconds)
Bit 0 LO ---- Retract Attack Mechanism

Action Byte							
R/W – 0	*R/W – 0*	*R/W – 0*	*R/W – 0*	*R/W – 0*	*R/W – 0*	*R/W – 0*	*R/W – 0*
--	--	--	--	--	--	--	**ACT0**
Bit 7							*Bit 0*

The ***Custom Bytes*** are bytes 4, 5 and 6 that follow the action byte. The functionality that results from these custom bytes varies from boat to boat. Each group is **required** to publish the functionality that these custom bytes provide for their boat. This allows any controller to execute the custom functionality of any boat. Byte value of **0x00** corresponds to no special action taken and thus should result in custom functionality to resolve to the default condition. When controlling a boat, each group must check the boat its controlling, and look up the custom functionality in a pre-assembled look-up table in order to implement said functionality.

Bytes 4 and Bytes 5 are required to be analog action byte, where the value **0x00** corresponds to no action. Analog action corresponds to actions such as aiming the popper or other form of analog functionality. The analog byte is a signed char ranging from -128 to 127.

Custom Byte 4							
R/W – 0	*R/W – 0*	*R/W – 0*	*R/W – 0*	*R/W – 0*	*R/W – 0*	*R/W – 0*	*R/W – 0*
CUS7	**CUS6**	**CUS5**	**CUS4**	**CUS3**	**CUS2**	**CUS1**	**CUS0**
Bit 7							*Bit 0*

Custom Byte 5							
R/W – 0	*R/W – 0*	*R/W – 0*	*R/W – 0*	*R/W – 0*	*R/W – 0*	*R/W – 0*	*R/W – 0*
CUS7	**CUS6**	**CUS5**	**CUS4**	**CUS3**	**CUS2**	**CUS1**	**CUS0**
Bit 7							*Bit 0*

Custom Byte 6 is a digital action, where the functionality corresponds to ON/OFF digital condition. Though there is only one digital custom byte, this allows for up to 8 different functions, one corresponding to each bit. The default no action value for the digital custom byte should be **0x00.**

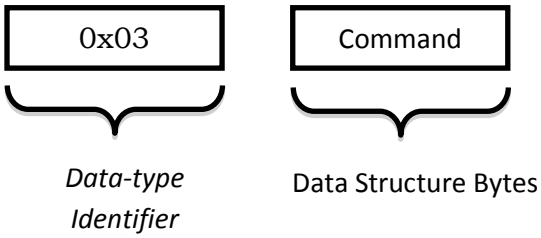
Custom Byte 6							
R/W – 0	*R/W – 0*	*R/W – 0*	*R/W – 0*	*R/W – 0*	*R/W – 0*	*R/W – 0*	*R/W – 0*
CUS7	**CUS6**	**CUS5**	**CUS4**	**CUS3**	**CUS2**	**CUS1**	**CUS0**
Bit 7							*Bit 0*

Bit HI ---- ON Condition
Bit LO ---- OFF Condition

Handshaking Procedure (0x03)

The handshaking data structure consists of the data-type identifier (0x03) followed by a command byte. The second data byte must be 0xED if the message is sent from the LUC. It must be 0xCD if the message is sent

from the HZV. Only the LUC may initiate the handshaking procedure. Handshaking can *only* be initiated with a button push on the LUC. The LUC will not be allowed to automatically resend the handshaking message without a button press if it does not receive an acknowledgement handshake message of 0x03 0xCD from the HZV.

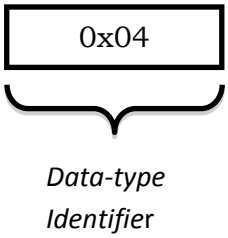


Command Logic Table

0xED ---- LUC to HZV command
0xCD ---- HZV to LUC acknowledgement

Nav/Action Acknowledgement (0x04)

The navigation/action acknowledgement data structure consists of the data-type identifier (0x04) only. The LUC will rely on receiving the acknowledgement byte to confirm control of the HZV.



Upon completing the handshaking process, the LUC will begin streaming navigation/action data at 5Hz and start a 1 second timer to receive the first acknowledgement. Upon each new acknowledgement, the 1-second timer will be restarted. If the 1-second timer expires, the LUC returns to a disconnected state. This could happen under 2 different circumstances. If the HZV's balloon has popped, it will stop sending acknowledgements. Also, if the XBees cannot make a connection, acknowledgements will not be received. After the handshaking process, the HZV will also start a 1 second timer for receiving LUC commands and restart the timer on each received packet. If the timer expires, the HZV will return to a disconnected state and be available for control. Due to the 1-second timer, the microcontrollers will not need to respond to a nack. If the radios cannot establish connection, the acknowledgement byte will never be sent and the LUC and HZV will both recognize that they are disconnected.