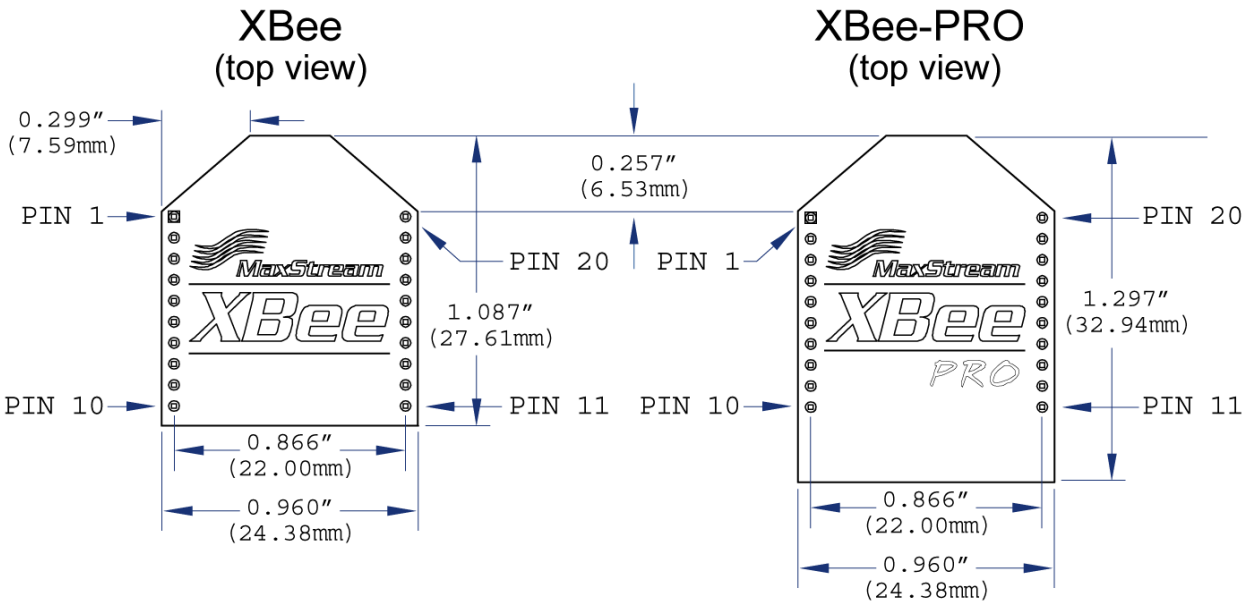
XBee Module Control for Droids

What is XBee:

XBee refers to the actual RF modules that you can use to implement wireless communications.

Some Physical Specifications:



Why XBee Modules?

XBee modules are a great way to add wireless control to your projects.

1. XBee modules are inexpensive starting at around $25.00.
2. Standard serial interface
3. Small form factor
4. Great range - up to 300 ft. (clear line of sight) for XBee and up for 1 mile (clear line of sight) for XBee Pro modules.
5. Low power for long battery life:
   < 50 mA when working hard
   < 10 uA when sleeping
6. Reliable communications
7. Implement either 900 MHz or 2.4 GHz
8. XBee modules offer some unique built in capabilities that would be difficult to implement with other technologies such as mesh networking.
9. XBee modules are easy to work with.
10. With 65,000 network addresses for each of 16 address channels you can have many of these in a network.

What is ZigBee?

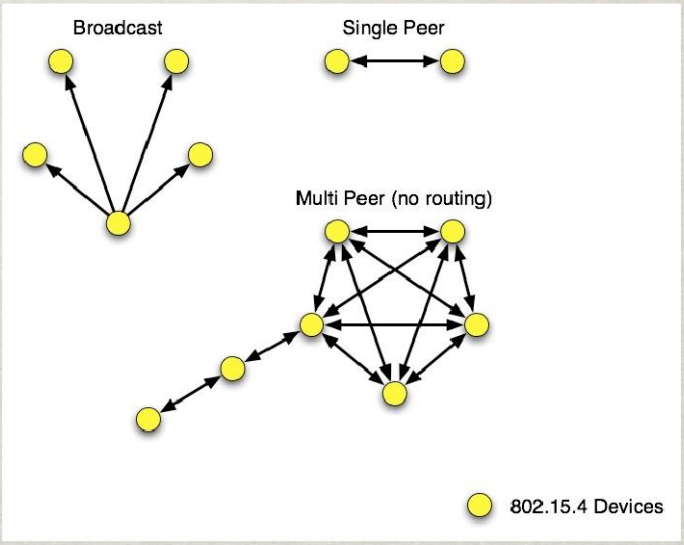
1. ZigBee is the communications protocol used by XBee modules.

2. ZigBee is built on top of the IEEE 802.15.4 protocol

How XBee's can communicate:

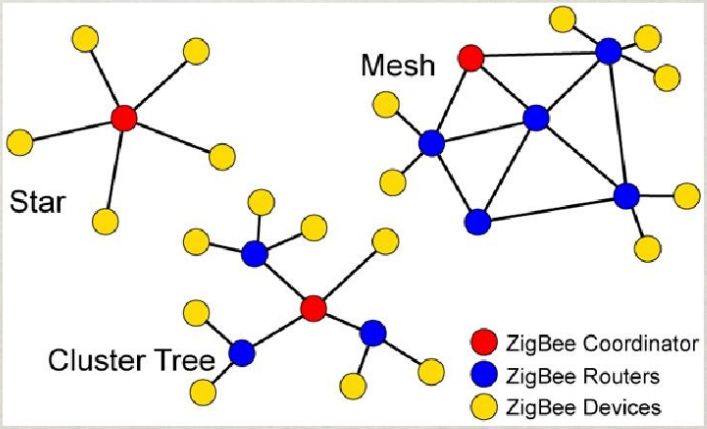
802.15.4 Topologies

- single peer
- multi-peer
- broadcast



ZigBee Topologies

- peer
- star
- mesh
- routing



XBee Modes of operation

XBees support two modes of operation:

AT or (transparent) or (UART) mode – Default setting
API or (frame) or (packet) mode – manually set

In AT (or transparent) mode, the XBee radio simply relays serial data to the receiving XBee, as identified by the DH+DL address.

In API mode, you communicate with the radio by sending and receiving structured packets of data.

Features of the Transparent (AT) Mode:

For creating a simple point-to-point configuration, where the XBee modules act as a wireless serial modem between a computer or microcontroller and a remote device using simple serial communications.

Features of the AT or (Transparent) mode:

- Simple
- Compatible with any device that speaks serial
- Limited to point to point communication between two XBees
- Primarily for point to point communication between two XBees. It's possible to communicate with multiple XBees but this requires entering command mode each time to change the destination address.

NOTE: AT mode is the default mode of the RF module.

NOTE: Series 1 radios support both AT and API modes with a single firmware version, allowing you switch between the modes with X-CTU. However, Series 2 requires a specific firmware for API mode. As of now there are two firmware versions for Series 2 API mode: ZNet and ZB Pro. ZNet is recommended as it is the easiest to work with.

Features of the API Mode:

In API mode, you communicate with the radio by sending and receiving structured packets of data. This requires XBees to be manually configured to be in API mode.

Features of the API mode:

- I/O Line passing (receive data from a stand-alone remote XBee)
- Allows for Broadcast communication and communication with more than one XBee
- Obtain RSSI (signal strength) of an RX packet
- Configure a remote radio with the Remote AT feature
- Allows an XBee to receive I/O data from 1 or more remote XBees Acknowledgement (ACK) and Retries.
- The transmitting radio will resend the packet if it does not receive an ACK.
- Receive packets (RX), contain the source address of transmitting radio
- Easily address multiple radios and send broadcast TX packets
- Packets include a checksum for data integrity
- Send 100 bytes in length for the XBee or up to 2048 bytes with the 9XTend in default configuration and API enabled.

NOTE: API mode is not the default mode and must be enabled by setting the AP parameter.

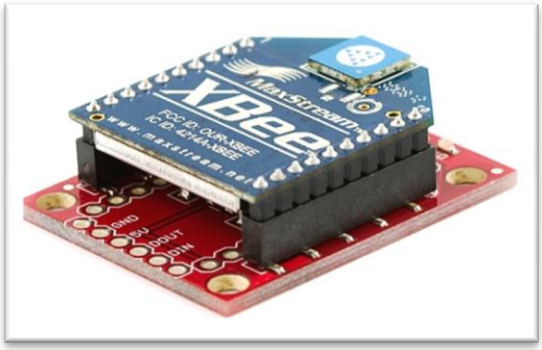
Getting Started

Here's the hardware and software you will need to set up the XBee radios before you can use them:

- **Hardware:** XBee series 1 RF module(s)
- Computer running Windows XP or more recent OS
- **Software:** FTDI Driver for XBee Explorer USB
- **Software:** X-CTU by Digi
  http://www.digi.com/support/kbase/kbaseresultdetl.jsp?kb=125
- **Hardware:** XBee Explorer USB - For the XBee attached to your computer via USB

http://www.sparkfun.com/products/8687

- **Hardware:** XBee Explorer Regulated - For the remote XBee module



http://www.sparkfun.com/products/9132

Setting up Your XBee(s)

• On your windows computer, install the FTDI drivers and X-CTU software

• Carefully plug the XBee into the XBee Explorer USB, making sure the angled sides are facing away from the USB port and matching the white lines drawn on the Explorer.

• Plug the USB cable into your computer.

• Start the X-CTU utility program

• With the PC Setting tab selected at the top of the X-CTU interface, select the appropriate COM port.

• First, update the firmware to 10CD or later.

• Assuming your XBees are new, use the default settings for the Baud Rate, Flow Control, etc. (i.e. 9600, NONE, 8, NONE, 1)

• Then set the individual settings for the local and remote XBee units

Configuration of the Coordinator (local) XBee

The Coordinator XBee is the module that's connected to your computer (or microcontroller) and is the master unit. It can send data to many remote XBees (in API mode) as well as receive data from any remote XBee.

Networking & Security Section:

- CH

  The channel number used for transmitting and receiving data between RF modules (uses 802.15.4 protocol channel numbers).

- ID=3332 (the default ID)

  The ID is called the PAN ID (Personal Area Network), and establishes a common network among all the XBees - Every XBee in your network MUST have the same number for the PAN ID (you can set it differently from 3332 if you want).
  Use 0xFFFF to broadcast messages to all PANs.
  All modules within a PAN should operate using the same firmware version.

- DH

  DH (Destination High) is the upper 32 bits of the 64-bit destination address. When combined with DL, it defines the destination address used for transmission. To transmit using a 16-bit address, set DH parameter to zero and DL less than 0xFFFF.

- DL

  DH (Destination Low) is the lower 32 bits of the 64-bit destination address. When combined with DH, DL defines the destination address used for transmission. To transmit using a 16-bit address, set DH parameter to zero and DL less than 0xFFFF.

- SH

  SH (Serial Number High) is the high 32 bits of the RF module's unique IEEE 64-bit address.

- SL

  SH (Serial Number Low) is the low 32 bits of the RF module's unique IEEE 64-bit address

- MY

  This is the remote RF modules 16-bit address, so each End-Device should have a unique value for MY. Note: This number is a HEX value, if you set MY=10, that is equal to 16 in the decimal system that you would use in the AnalogInput widget for the xbeeRemoteID. Set MY = 0xFFFF to disable reception of packets with 16-bit addresses.

- CE

  The Coordinator Enable Setting
  CE=1 (A Coordinator Module)
  CE=0 (An End Device or Receiver Module)

Serial Interfacing Section:

- BD

  The serial interface data rate for communications between the RF module serial port and host. Remember you will have to set X-CTU and any other software to communicate at that rate once you've changed it.

- AP

  Disable/Enable API Mode Setting
  0=API Disabled
  1=API enabled
  2=API enabled (w/escaped control characters)

- IT

  Samples before TX - how many samples the XBee will collect before transmitting them back to the coordinator.

- IR

  Sample Rate in milliseconds - i.e. the XBee will transmit the number of IT samples every 64 milliseconds (about 15 times a second). You can make the Sample Rate lower so it transmits more frequently, but that will use more

power because it is transmitting more often. Set the number higher (i.e. fewer times per second)

API Mode Operation
(Application Programming Interface)

In API mode, all data entering and leaving the module is contained in frames that cause operations or events within the module.

All UART data received through the DI pin is queued up for RF transmission.

When the module receives an RF packet, the data is sent out the DO pin with no additional information.

Frames received through pin 3 (DI) are called Transmit Data Frames. Frames sent out through pin 2 (DO) are called Receive Data Frames.

API operation requires that communication with the module be done through a structured interface (data is communicated in frames in a defined order).

The API specifies how commands, command responses and module status messages are sent and received from the module using a UART Data Frame.

Two API modes are supported and both can be enabled using the AP (API Enable) command.

Use the following AP parameter values to configure the module to operate in a particular mode:

• AP = 0 (default): Transparent Operation (UART Serial line replacement) API modes are disabled.

• AP = 1: API Operation

• AP = 2: API Operation (with escaped characters) Any data received prior to the start delimiter is silently discarded. If the frame is not received correctly or if the checksum fails, the data is silently discarded. Operation with escaped control characters would be necessary if using XON/XOFF flow control.

API Packets

When operating in API mode, all communication between the host and the XBee is by means of these binary packets.
There are thirteen packet types. There are nine main types, and four of those have two sub-types.

Packets Types you can send:

Packet Type	
0x00	TX request with 64-bit destination address
0x01	TX request with 16-bit destination address
0x08	Local AT command, immediate action
0x09	Local AT command, queued action
0x17	Remote AT command with 64-bit destination address

Packets you may receive:

Packet Type	
0x80	RX with 64-bit source address
0x81	RX with 16-bit source address
0x82	Input line states with 64-bit source address
0x83	Input line states with 16-bit source address
0x88	Local AT response
0x89	TX response
0x8a	Modem status packet
0x97	Remote AT response

All Packets:

Every packet has a three-byte header, a payload and a single-byte checksum:

• The first byte of the header (Also known as the Start Delimiter byte) is always 0x7e.

• The second byte of the header is the most significant byte of the payload length. It is always zero, because currently there are no packets longer than 255 bytes.

• The third byte of the header is the least significant byte of the payload length.

• The payloads for the different packet types are described below. Every payload begins with a byte which identifies It's packet type.

• The checksum is calculated from the payload bytes only. Its value is the sum of the payload bytes, subtracted from 0xff. When checking the checksum on an incoming packet, the sum of the payload bytes plus the checksum byte should be 0xff. All calculations are done modulo 0x100.

API Packet Structure

API Operation (AP parameter = 1)

When this API mode is enabled (AP = 1), the UART data frame structure is defined as follows:

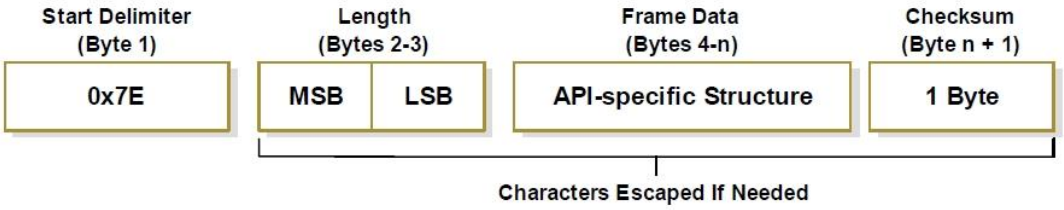
Figure 3-01. UART Data Frame Structure:

MSB = Most Significant Byte, LSB = Least Significant Byte

API Operation - with Escape Characters (AP parameter = 2)

When this API mode is enabled (AP = 2), the UART data frame structure is defined as follows:

Figure 3-02. UART Data Frame Structure - with escape control characters:



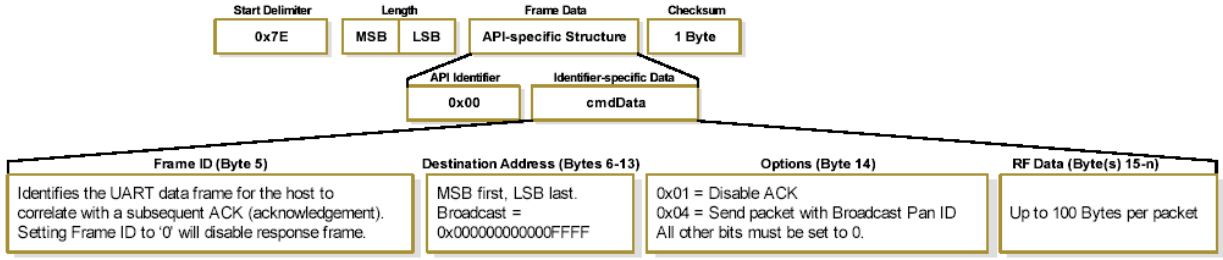
MSB = Most Significant Byte, LSB = Least Significant Byte

TX Transmit Request Frame Data Structure

TX (Transmit) Request: 64-bit address

API Identifier Value: 0x00
A TX Request message will cause the module to send RF Data as an RF Packet.

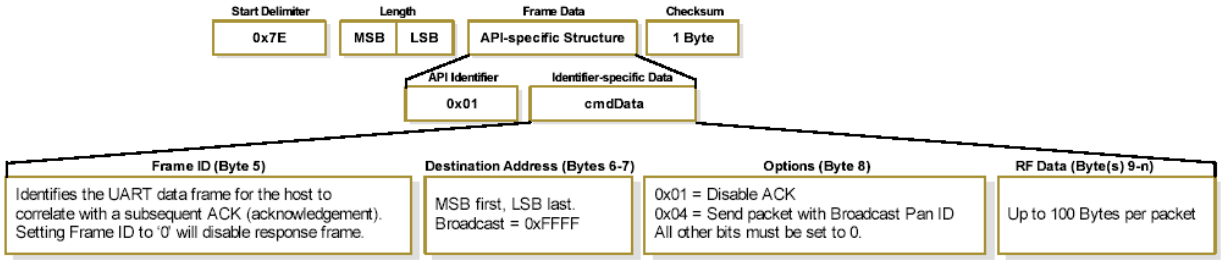
Figure 3-10. TX Packet (64-bit address) Frames



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



API Mode Examples:

This example will send the ASCII string "Hello" to a radio with the destination address of 5001. Both examples will be using API mode without escaped characters (ATAP=1). Both API options within the radios became readily available to customers on firmware version 1083 for the XBee and 2020 for the 9XTend. For the ZigBee versions of the firmware, please verify you have install the API version.

XBee - 16 bit 802.15.4 Unicast Example:

Configure the radios with the following parameters:

Radio 1	Radio 2
AP = 1	DL = 5000
MY = 5000	MY = 5001

To compose the data packet, use the ''Assemble Packet'' option found within the Terminal tab of the X-CTU test and configuration software.

Launch a second X-CTU screen. Verify you have selected the COM port Radio 2 is on, and click on the Terminal tab.

Select the ''Hex'' option in the ''Assemble Packet'' window and enter the following hex formatted data in the X-CTU set for Radio 1's COM port:

7E 00 0A 01 01 50 01 00 48 65 6C 6C 6F B8

7E	Start delimiter
00 0A	Length bytes
01	API identifier
01	ACK Byte – 0 = No response
50 01	Destination address low
00	Option byte
48 65 6C 6C 6F	Data packet
B8	Checksum

If the packet was successful, you will see "Hello" in the receiving modules Terminal window. You will also receive a response packet back to Radio 1 stating the packet was successful. This successful response packet will look something like this:

7E 00 03 89 01 00 75

7E	Start delimiter
00 03	Length bytes
89	API identifier
01	API frame ID
00	Status byte
75	Checksum

To view the TX response packet, use the ''Show Hex'' option in the Terminal tab

All other parameters are unchanged. Source and Destination Addresses were arbitrarily chosen.

*Radio 2's AP option can be set to either 0 (API off), 1(API without escaped characters) or 2 (with escaped characters). The receiving message will still be the same.

The radios are ready to communicate the "Hello" message using an API packet. Select the X-CTU window you have selected as Radio 1 and click on the Terminal tab.

Select the ''Hex'' option in the ''Assemble Packet'' window and enter the following hex formatted data:

7E 00 0A 01 01 50 01 00 48 65 6C 6C 6F B8

7E	Start delimiter
00 0A	Length bytes
01	API identifier
01	API frame ID
50 01	Destination address low

00	Option byte
48 65 6C 6C 6F	Data Packet
B8	Checksum

The 9XTend radio with the address of 5001 (Radio 2) will receive the "Hello" data packet and will be displayed in the Terminal tab of its corresponding X-CTU. The transmit response on Radio 1's packet will now look like:

7E 00 03 89 01 00 75

7E	Start delimiter
00 03	Length bytes
89	API identifier
01	API frame ID
00	Status byte
75	Checksum

My XBee Modules:

Coordinator:

CH=C	Channel
PANID=8649	Personal Area Network ID
DH=0	Destination Address High
DL=FFFF	Destination Address Low
SH=13A200	Serial Number High - part of the 64 bit serial number
SL=406A9A35	Serial Number Low - part of the 64 bit serial number
MY=1234	Source Address

CE=1-coordinator

BD=1-2400	Baud Rate
AP=1 - API Enabled	Transmission mode

Receiver:

CH=C	Channel
PANID=8649	Personal Area Network ID
DH=0	Destination Address High
DL=3141	Destination Address Low
SH=13A200	Serial Number High - part of the 64 bit serial number
SL=4063B15A	Serial Number Low - part of the 64 bit serial number
MY=4321	Source Address

CE=0 receiver

BD=1-2400	Baud Rate
AP=2 - API Enabled	Transmission mode

Packet: 7E 00 0A 01 14 FF FF 03 48 65 6C 6C 6F F5

7E 1 Byte - Packet header byte (always 7E) - this is the XBee frame delimiter, indicatingthe start of a new API packet
00 0A 2 Bytes - payload length (in this case = 10 bytes not including the checksum byte = 01, 14, FF, FF, 03, 48, 65, 6C, 6C, 6F - each number is one byte)

*Checksum calculation would start here
01 1 Byte - Start of Frame data (API Identifier or Packet type) = TX request with 16-bit address
14 1 Byte - frame id: non-zero so response packet expected, if 0 then no response
FF FF 2 Bytes - 16-bit destination address, FFFF broadcasts to all modules on the network
03 1 Byte - TX options: Disable ACK, Address broadcast

Transmitted data begins next. In this packet (5 bytes) but can be up to 100 Bytes

48 1 - Byte - transmitted data is a upper case H
65 1 - Byte - transmitted data is a lower case e - This is the hexidecimal value for the ASCII value 'e'
6C 1 - Byte - transmitted data is a lower case l
6C 1 - Byte - transmitted data is a lower case l
6F 1 - Byte - transmitted data is a lower case o
*Checksum calculation ends here

F5 1 - Byte - checksum

To test data integrity, a checksum is calculated and verified on non-escaped data.

To calculate:
Do Not include frame delimiters and length (the first 3 Bytes of the frame) or the ending checksum.
Add all bytes keeping only the lowest 8 bits of the result and subtract from 0xFF.

Original frame data: 7E 00 0A 01 14 FF FF 03 48 65 6C 6C 6F F5

01 14 FF FF 03 48 65 6C 6C 6F = 40A in Hex and 1034 in decimal

Take the last byte from 40A which is 0A
Subtract from FF
FF - 0A = F5

To verify:
Add all bytes (include checksum, but not the delimiter and length).
If the checksum is correct, the sum will equal 0xFF.

Creating and API mode packet:

Sending data in API mode is not so simple. Before further discussion, it must be added that API mode has two sub-modes: escaped and un-escaped. Escaped mode requires that any data that can conflict with API control-characters must be preceded with 0x7D and XOR'd with 0x20. That procedure ensures that the XBee module won't be confused with any user data that happens to be same as any of the XBee control characters. This example will use escaped mode. It has the added benefit of making it easier to program receive routines.

Let's revisit our earlier example of sending the byte 0x7E. All that was required to send a byte was a single function call. In API mode several packet formatting steps must first take place:

1. Create a temporary buffer guaranteed to be bigger than needed

2. Insert 0x7E into position 0, this is the XBee frame delimiter, indicating the start of a new API packet00x7E

3. Calculate the length of the packet, which in our case will be several bytes longer than the single 0x7E we want to send. Put the upper8 bits of that length into buffer position 1, and the lower 8 bits into position 2120x00 0x0F Engineering 428—Kickliter34. Add the XBee API identifier to position

4. There are several ID's for different purposes, but we are trying to send a packet to another XBee, so we will use 0x10, which is the TX Request id 30x10

5. Insert a unique frame ID byte into position 4. This is not very important unless you want the module to respond with an acknowledge packet that it received the transmit request, however a byte needs to be here, so we'll use 0x0140x01

6. Insert the 64 bit address of the remote module into positions 5–12, send most significant byte first

7. Insert the 16 bit address of the network you are operating on into positions 13–14, MSB first. We'll use the address 0xFFFE, since it is the default

8. Insert 0x00 into both position 15 and 16, these fields have definitions, but are not flexible, and serve no real purpose

9. Finally, staring at position 17, insert however many bytes you need to send, in our case, just insert 0x7E into position 17

10. Calculate the checksum of the packet and insert it into position n, which in our case is 18. To calculate checksum, we add all the Bytes from positions 3 to n-1, discarding any carry and

keeping only the bottom 8 bits. Subtract this number from 0xFF. What is left over is our checksum.

Checksum = 0xFF - ((0x00 + 0x01 + 0x00 + 0x13 + 0xA2 + 0x00 +0x40 + 0x4B + 0x22 + 0x77 + 0xFF + 0xFE + 0x00 + 0x00 +0x7E) & 0xFF)

Checksum = 0xAA11. Our (almost) completed packet looks like:
01234567890x7E 0x00 0x0F 0x10 0x01 0x00 0x13 0xA2 0x00 0x4010111213141516171800x4B 0x22 0x77 0xFF 0xFE 0x00 0x00 0x7E 0xAA12.

11. Starting at byte 1, we must enumerate through the whole packet and escape any control characters. Any byte with the vale 0x7E, 0x7D,0x11, 0x13 must be replaced with 0x7D, and be followed by that byte's value XOR'd with 0x20. Since our packet has a 0x13 at index 6, and a0x7E at index 17, we will have to do two escapes.

After escaping those two bytes, our packet will look like:
012345670x7E 0x00 0x0F 0x10 0x01 0x00 0x7D 0x338910111213140xA2 0x00 0x40 0x4B 0x22 0x77 0xFF1516171819200xFE 0x00 0x00 0x7D 0x5E 0xAA13.

12. Lastly, all that is left to send the packet is to transmit bytes to the XBee module over the serial line

Useful Links:

http://www.makingthings.com/documentation/tutorial/xbee-wireless-interface
http://www.faludi.com/itp_coursework/meshnetworking/XBee/