XBee® / XBee-PRO® ZB SMT RF Modules

ZigBee RF Modules by Digi International

Models: XBEE S2C, PRO S2C

Hardware: S2C

Firmware: 401x, 402x, 403x, 404x

Digi International Inc.
11001 Bren Road East
Minnetonka, MN 55343
877 912-3444 or 952 912-3444
http://www.digi.com

90002002_J
6/5/2013

Technical Support:	Phone:	(866) 765-9885 toll-free U.S.A. & Canada (801) 765-9885 Worldwide 8:00 am - 5:00 pm [U.S. Mountain Time]
	Live Chat:	www.digi.com
	Online Support:	http://www.digi.com/support/eservice/login.jsp
	Email:	rf-experts@digi.com

Contents

Contents

Contents

1. Overview

This manual describes the operation of the XBee/XBee-PRO ZB SMT RF module, which consists of ZigBee firmware loaded onto XBee S2C and PRO S2C hardware.

XBee® and XBee-PRO® ZB SMT embedded RF modules provide wireless connectivity to end-point devices in ZigBee mesh networks. Utilizing the ZigBee PRO Feature Set, these modules are interoperable with other ZigBee devices, including devices from other vendors. With the XBee®, users can have their ZigBee network up-and-running in a matter of minutes without configuration or addtional development.

The XBee®/XBee-PRO® ZB modules are compatible with other devices that use XBee® "ZB" technology. These include ConnectPortX gateways, XBee® and XBee-PRO® Adapters, Wall Routers, XBee® Sensors, and other products with the "ZB" name.

Worldwide Acceptance

FCC Approval (USA) Refer to Appendix A for FCC Requirements. Systems that contain XBee®/ XBee-PRO® ZB SMT RF Modules inherit Digi Certifications.

ISM (Industrial, Scientific & Medical) 2.4 GHz frequency band

Manufactured under ISO 9001:2000 registered standards

XBee®/XBee-PRO® SMT ZB RF Modules are optimized for use in US, Canada, Europe, Australia, and Japan (contact Digi for complete list of agency approvals).

What's New in 40xx Firmware

- An alternative serial port is available using SPI slave mode operation.
- Six software images (Coordinator AT, Coordinator API, Router AT, Router API, End Device AT, and End Device API) are combined into a single software.
- Fragmentation is now available in both API mode and transparent mode.
- P3 (DOUT), P4 (DIN), D8 (SleepRq), and D9 (On-$\overline{\text{Sleep}}$) are now available for I/O sampling.
- Both pull-up and pull-down resistors can now be applied to pins configured for inputs.
- 401D - ATVL command added for long version information
- 401E - ATDO command added for configuring device options
- 4020 - ATAS command added for Active Scan
- 4021 - Self addressed Tx Status messages return a status code of 0x23
- ATDO has HIGH_RAM_CONCENTRATOR and NO_ACK_IO_SAMPLING options added.
- 4040 - Binding and Multicasting transmissions are supported.
- AT&X command added to clear binding and group tables.
- Added Tx options 0x04 (indirect addressing) and 0x08 (multicast addressing).
- A 5 second break will reset the XBee. Then it will boot with default baud settings into command mode.

Specifications

Specifications of the XBee®/XBee-PRO® ZB SMT RF Module

Specification	XBee	XBee-PRO
Performance		
Indoor/Urban Range	Up to 200 ft. (60 m)	Up to 300 ft. (90 m)
Outdoor RF line-of-sight Range	Up to 4000 ft. (1200 m)	Up to 2 miles (3200 m)
Transmit Power Output	6.3mW (+8dBm), Boost mode 3.1mW (+5dBm), Normal mode Channel 26 max power is +3dBm	63mW (+18 dBm, adjustable to 0 dBm)
RF Data Rate	250,000 bps	
Receiver Sensitivity	-102 dBm, Boost mode -100 dBm, Normal mode	-101 dBm
Power Requirements		
Adjustable Power	Yes	
Supply Voltage	2.1 - 3.6 V 2.2 - 3.6 V for Programmable Version	2.7 - 3.6 V
Operating Current (Transmit))	45mA (+8 dBm, Boost mode) 33mA (+5 dBm, Normal mode)	100mA @ +3.3 V, +18 dBm
Operating Current (Receive)	31mA (Boost mode) 28mA (Normal mode)	31mA
Power-down Current	< 1 μA @ 25^{o}C	
General		
Operating Frequency Band	ISM 2.4 - 2.5 GHz	
Dimensions	0.866” x 1.33” x 0.120” (2.199cm x 3.4cm x 0.305cm)	
Weight	1.4 oz. (40 g)	
Operating Temperature	-40 to 85º C (industrial)	
Antenna Options	RF Pad, PCB Antenna, or U.FL Connector	
Networking & Security		
Supported Network Topologies	Point-to-point, Point-to-multipoint, Peer-to-peer, and Mesh	
Number of Channels	16 Direct Sequence Channels	15 Direct Sequence Channels
Interface Immunity	DSSS (Direct Sequence Spread Spectrum)	
Channels	11 to 26	11 to 25
Addressing Options	PAN ID and Addresses, Cluster IDs and Endpoints (optional)	
Interface Options		
UART	1 Mbps maximum (burst)	
SPI	5 Mbps maximum (burst)	
Agency Approvals		
United States (FCC Part 15.247)	FCC ID: MCQ-XBS2C	FCC ID: MCQ-XBPS2C
Industry Canada (IC)	IC: 1846A-XBS2C	IC: 1846A-XBPS2C
Europe (CE)	ETSI	
Australia	C-Tick	

Specifications of the XBee®/XBee-PRO® ZB SMT RF Module

Specification	XBee	XBee-PRO
Japan	R201WW10215369	
RoHS	Compliant	

Serial Communications Specifications

XBee RF modules support both UART (Universal Asynchronous Receiver / Transmitter) and SPI (Serial Peripheral Interface) serial connections.

UART

The SC1 (Serial Communication Port 1) of the Ember 357 is connected to the UART port.

UART Pin Assignments

UART Pins	Module Pin Number
DOUT	3
DIN / $\overline{\text{CONFIG}}$	4
$\overline{\text{CTS}}$ / DIO7	25
$\overline{\text{RTS}}$ / DIO6	29

More information on UART operation is found in the UART section in Chapter 2.

SPI

The SC2 (Serial Communication Port 2) of the Ember 357 is connected to the SPI port.

SPI Pin Assignments

SPI Pins	Module Pin Number
SPI_SCLK / DIO18	14
SPI_$\overline{\text{SSEL}}$ / DIO17	15
SPI_MOSI / DIO16	16
SPI_MISO / DIO15	17

For more information on SPI operation, see the SPI section in Chapter 2.

GPIO Specifications

XBee RF modules have 15 GPIO (General Purpose Input / Output) ports available. The exact list will depend on the module configuration, as some GPIO pads are used for purposes such as serial communication.

See GPIO section for more information on configuring and using GPIO ports.

Electrical Specifications for GPIO Pads

GPIO Electrical Specification	Value
Voltage - Supply	2.1 - 3.6 V
Low Schmitt switching threshold	0.42 - 0.5 x VCC
High Schmitt switching threshold	0.62 - 0.8 x VCC
Input current for logic 0	-0.5 µA
Input current for logic 1	0.5 µA
Input pull-up resistor value	29 kΩ
Input pull-down resistor value	29 kΩ
Output voltage for logic 0	0.18 x VCC (maximum)

Electrical Specifications for GPIO Pads

GPIO Electrical Specification	Value
Output voltage for logic 1	0.82 x VCC (minimum)
Output source current for pad numbers 3, 4, 5, 10, 12, 14, 15, 16, 17, 25, 26, 28, 29, 30, and 32	4 mA
Output source current for pad numbers 3, 4, 5, 10, 12, 14, 16, 17, 26, 28, 29, 30, and 33	4 mA
Output source current for pad numbers 7, 8, 24, 31, and 33	8 mA
Output sink current for pad numbers 7, 8, 24, 31, and 33	8 mA
Total output current (for GPIO pads)	40 mA

Hardware Specs for Programmable Variant

If the module has the programmable secondary processor, add the following table values to the specifications listed on page 7. For example, if the secondary processor is running at 20 MHz and the primary processor is in recieve mode then the new current value will be $I_{total} = I_{r2} + I_{rx}$ = 14 mA + 9 mA = 23 mA, where I_{r2} is the runtime current of the secondary processor and I_{rx} is the recieve current of the primary.

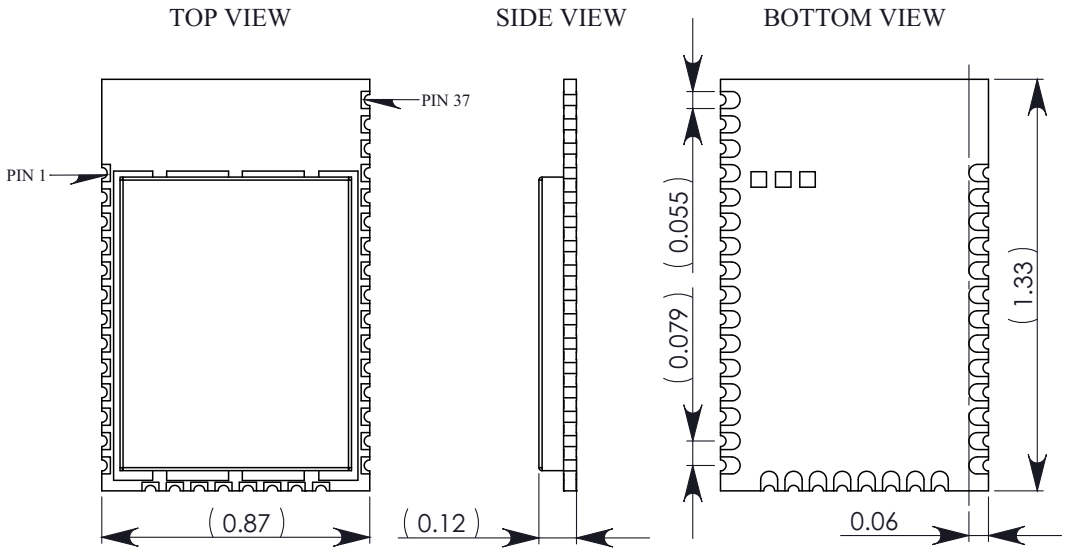
Specifications of the programmable secondary processor

Optional Secondary Processor Specification	These numbers add to specifications (Add to RX, TX, and sleep currents depending on mode of operation)
Runtime current for 32k running at 20MHz	+14mA
Runtime current for 32k running at 1MHz	+1mA
Sleep current	+0.5µA typical
For additional specifications see Freescale Datasheet and Manual	MC9SO8QE32
Minimum Reset low pulse time for EM357	+26µS
VREF Range	1.8VDC to VCC

Mechanical Drawings

Mechanical drawings of the XBee®/XBee-PRO® ZB SMT RF Modules (antenna options not shown). All dimentions are in inches.

.



Pin Signals

Pin Assignments for XBee Modules

(Low-asserted signals are distinguished with a horizontal line above signal name.)

Pin #	Name	Direction	Default State	Description
1	GND	-	-	Ground
2	VCC	-	-	Power Supply
3	DOUT / DIO13	Both	Output	UART Data Out / GPIO
4	DIN / $\overline{\text{CONFIG}}$ / DIO14	Both	Input	UART Data In / GPIO
5	DIO12	Both		GPIO
6	$\overline{\text{RESET}}$	Input		Module Reset
7	RSSI PWM / DIO10	Both	Output	RX Signal Strength Indicator / GPIO
8	PWM1 / DIO11	Both	Disabled	Pulse Width Modulator / GPIO
9	[reserved]	-	Disabled	Do Not Connect
10	$\overline{\text{DTR}}$ / SLEEP_RQ / DIO8	Both	Input	Pin Sleep Control Line / GPIO
11	GND	-	-	Ground
12	SPI_$\overline{\text{ATTN}}$ / $\overline{\text{BOOTMODE}}$ / DIO19	Output	Output	Serial Peripheral Interface Attention Do not tie low on reset
13	GND	-	-	Ground
14	SPI_CLK / DIO18	Input	Input	Serial Peripheral Interface Clock / GPIO
15	SPI_$\overline{\text{SSEL}}$ / DIO 17	Input	Input	Serial Peripheral Interface not Select / GPIO
16	SPI_MOSI / DIO16	Input	Input	Serial Peripheral Interface Data In / GPIO
17	SPI_MISO / DIO15	Output	Output	Serial Peripheral Interface Data Out / GPIO
18	[reserved]*	-	Disabled	Do Not Connect
19	[reserved]*	-	Disabled	Do Not Connect
20	[reserved]*	-	Disabled	Do Not Connect
21	[reserved]*	-	Disabled	Do Not Connect
22	GND	-	-	Ground
23	[reserved]	-	Disabled	Do Not Connect
24	DIO4	Both	Disabled	GPIO
25	$\overline{\text{CTS}}$ / DIO7	Both	Output	Clear to Send Flow Control / GPIO
26	ON / $\overline{\text{SLEEP}}$ / DIO9	Both	Output	Module Status Indicator / GPIO
27	VREF	Input	-	Not used for EM357. Used for programmable secondary processor. For compatibility with other XBee modules, we recommend connecting this pin to the voltage reference if Analog Sampling is desired. Otherwise, connect to GND.
28	ASSOCIATE / DIO5	Both	Output	Associate Indicator / GPIO
29	$\overline{\text{RTS}}$ / DIO6	Both	Input	Request to Send Flow Control / GPIO
30	AD3 / DIO3	Both	Disabled	Analog Input / GPIO
31	AD2 / DIO2	Both	Disabled	Analog Input / GPIO
32	AD1 / DIO1	Both	Disabled	Analog Input / GPIO
33	AD0 / DIO0	Both	Input	Analog Input / GPIO
34	[reserved]	-	Disabled	Do Not Connect
35	GND	-	-	Ground
36	RF	Both	-	RF IO for RF Pad Variant
37	[reserved]	-	Disabled	Do Not Connect

- Signal Direction is specified with respect to the module
- See Design Notes section below for details on pin connections.
- * Refer to the Writing Custom Firmware section for instructions on using these pins if JTAG functions are needed.

EM357 Pin Mappings

The following table shows how the EM357 pins are used on the XBee.

EM357 Pin #	EM357 Pin Name	XBee Pin #	Other Usage
12	$\overline{\text{RST}}$	6	Programming
18	PA7	8	
19	PB3	29	Used for UART
20	PB4	25	Used for UART
21	PA0 / SC2MOSI	16	Used for SPI
22	PA1 / SC2MISO	17	Used for SPI
24	PA2 / SC2SCLK	14	Used for SPI
25	PA3 / SC2$\overline{\text{SSEL}}$	15	Used for SPI
26	PA4 / PTI_EN	32	OTA packet tracing
27	PA5 / PTI_DATA / BOOTMODE	12	OTA pacet tracing, force embedded serial bootloader, and SPI attention line
29	PA6	7	
30	PB1 / SC1TXD	3	Used for UART
31	PB2 / SC1RXD	4	Used for UART
33	PC2 / JTDO / SWO	26	JTAG (see Writing Custom Firmware section)
34	PC3 / JTDI	28	JTAG (see Writing Custom Firmware section)
35	PC4 / JTMS / SWDIO	5	JTAG (see Writing Custom Firmware section)
36	PB0	10	
38	PC1 / ADC3	30	
41	PB7 / ADC2	31	
42	PB6 / ADC1	33	
43	PB5 / ADC0		Temperature sensor on PRO version

NOTE: Some lines may not go to the external XBee pins in the programmable secondary processor version.

Design Notes

The XBee modules do not specifically require any external circuitry or specific connections for proper operation. However, there are some general design guidelines that are recommended for help in troubleshooting and building a robust design.

Power Supply Design

Poor power supply can lead to poor radio performance, especially if the supply voltage is not kept within tolerance or is excessively noisy. To help reduce noise, we recommend placing both a 1µF and 8.2pF capacitor as near to pin 2 on the PCB as possible. If using a switching regulator for your power supply, switching frequencies above 500kHz are preferred. Power supply ripple should be limited to a maximum 250mV peak to peak.

Note – For designs using the programmable modules, an additional 10µF decoupling cap is recommended near pin 2 of the module. The nearest proximity to pin 2 of the three caps should be in the following order: 8.2pf, 1µF followed by 10µF.

Recommended Pin Connections

The only required pin connections are VCC, GND, DOUT and DIN. To support serial firmware updates, VCC, GND, DOUT, DIN, RTS, and DTR should be connected.

All unused pins should be left disconnected. All inputs on the radio can be pulled high or low with 30k internal pull-up or pull-down resistors using the PR and PD software commands. No specific treatment is needed for unused outputs.

For applications that need to ensure the lowest sleep current, unconnected inputs should never be left floating. Use internal or external pull-up or pull-down resistors, or set the unused I/O lines to outputs.

Other pins may be connected to external circuitry for convenience of operation, including the Associate LED pad (pad 28) and the Commissioning pad (pad 33). The Associate LED pad will flash differently depending on the state of the module to the network, and a pushbutton attached to pad 33 can enable various join functions without having to send serial port commands. Please see the commissioning pushbutton and associate LED section in chapter 7 for more details. The source and sink capabilities are limited to 4mA for pad numbers 3, 4, 5, 10, 12, 14, 15, 16, 17, 25, 26, 28, 29, 30 and 32, and 8mA for pad numbers 7, 8, 24, 31 and 33 on the module.

The VRef pad (pad 27) is only used on the programmable versions of these modules. For compatibility with other XBee modules, we recommend connecting this pin to a voltage reference if analog sampling is desired. Otherwise, connect to GND.

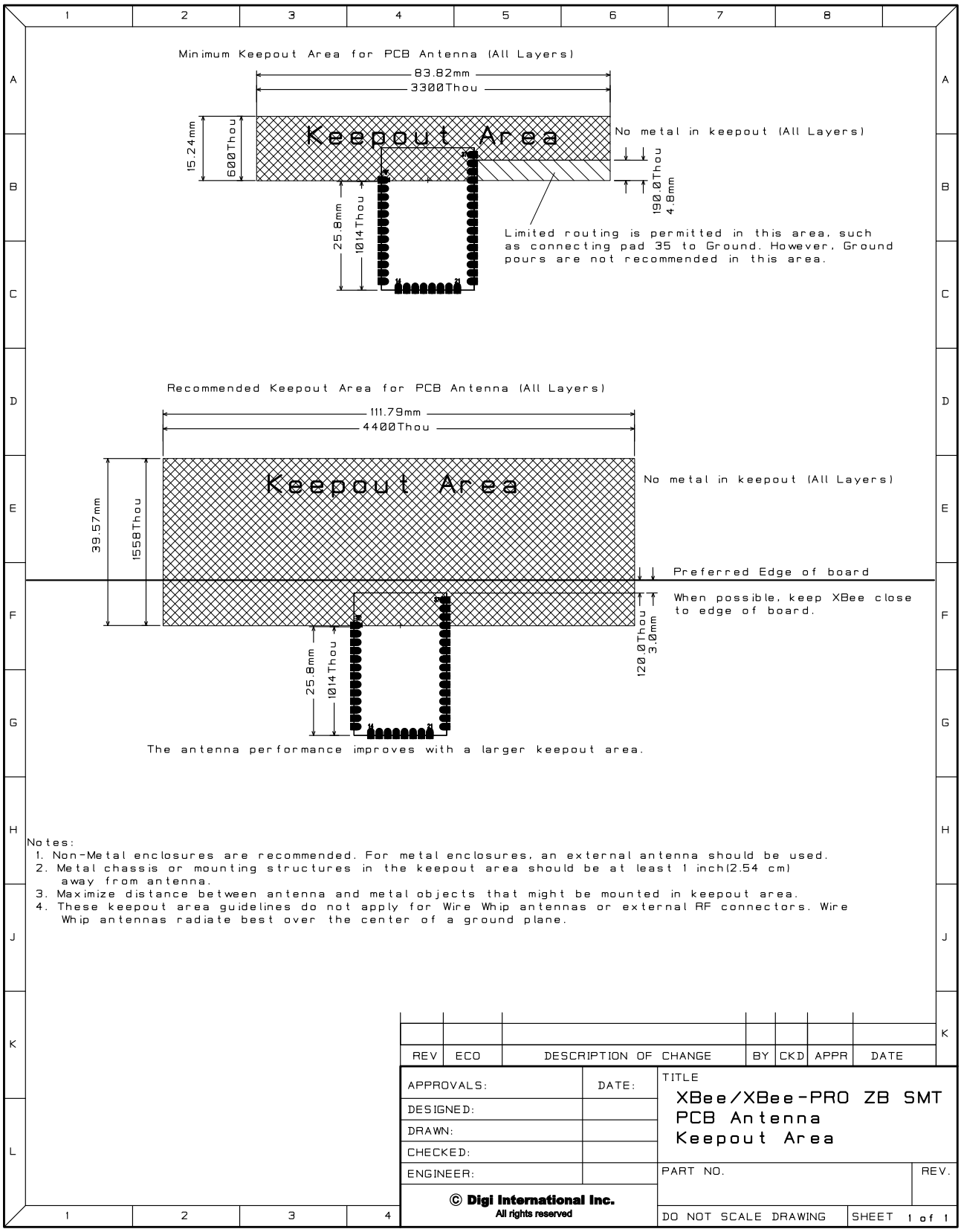
Board Layout

XBee modules are designed to be self sufficient and have minimal sensitivity to nearby processors, crystals or other PCB components. As with all PCB designs, Power and Ground traces should be thicker than signal traces and able to comfortably support the maximum current specifications. A recommended PCB footprint for the module can be found in Appendix C. No other special PCB design considerations are required for integrating XBee radios except in the antenna section.

The choice of antenna and antenna location is very important for correct performance. With the exception of the RF Pad variant, XBees do not require additional ground planes on the host PCB. In general, antenna elements radiate perpendicular to the direction they point. Thus a vertical antenna emits across the horizon. Metal objects near the antenna cause reflections and may reduce the ability for an antenna to radiate efficiently. Metal objects between the transmitter and receiver can also block the radiation path or reduce the transmission distance, so external antennas should be positioned away from them as much as possible. Some objects that are often overlooked are metal poles, metal studs or beams in structures, concrete (it is usually reinforced with metal rods), metal enclosures, vehicles, elevators, ventilation ducts, refrigerators, microwave ovens, batteries, and tall electrolytic capacitors.

Design Notes for PCB Antenna Modules

PCB Antenna modules should not have any ground planes or metal objects above or below the antenna. For best results, the module should not be placed in a metal enclosure, which may greatly reduce the range. The module should be placed at the edge of the PCB on which it is mounted. The ground, power and signal planes should be vacant immediately below the antenna section. The drawing on the following page illustrates important recommendations for designing with the PCB Antenna module. It should be noted that for optimal performance, this module should not be mounted on the RF Pad footprint described in the next section because the footprint requires a ground plane within the PCB Antenna keep out area.

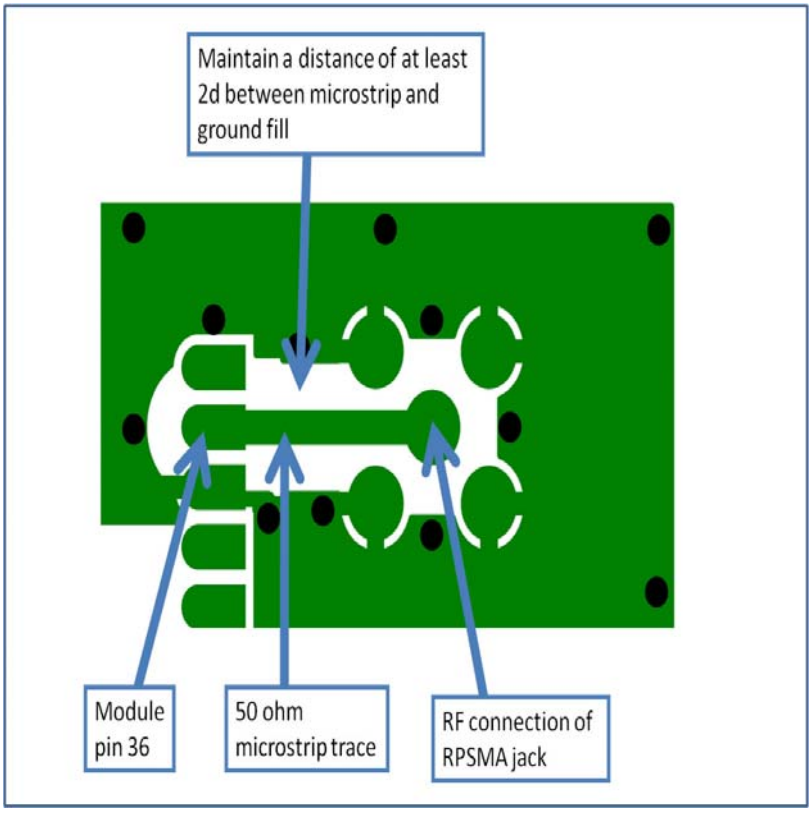

Minimum Keepout Area for PCB Antenna (All Layers)
83.82mm
3300Thou
Keepout Area
15.24mm
600Thou
No metal in keepout (All Layers)
190.0Thou
4.8mm
25.8mm
1014Thou
Limited routing is permitted in this area, such as connecting pad 35 to Ground. However, Ground pours are not recommended in this area.
Recommended Keepout Area for PCB Antenna (All Layers)
111.79mm
4400Thou
Keepout Area
39.57mm
1558Thou
No metal in keepout (All Layers)
Preferred Edge of board
When possible, keep XBee close to edge of board.
120.0Thou
3.0mm
25.8mm
1014Thou
The antenna performance improves with a larger keepout area.
Notes:
1. Non-Metal enclosures are recommended. For metal enclosures, an external antenna should be used.
2. Metal chassis or mounting structures in the keepout area should be at least 1 inch(2.54 cm) away from antenna.
3. Maximize distance between antenna and metal objects that might be mounted in keepout area.
4. These keepout area guidelines do not apply for Wire Whip antennas or external RF connectors. Wire Whip antennas radiate best over the center of a ground plane.
REV
ECO
DESCRIPTION OF CHANGE
BY
CKD
APPR
DATE
APPROVALS:
DATE:
DESIGNED:
DRAWN:
CHECKED:
ENGINEER:
TITLE
XBee/XBee-PRO ZB SMT PCB Antenna Keepout Area
PART NO.
REV.
© Digi International Inc.
All rights reserved
DO NOT SCALE DRAWING
SHEET 1 of 1

Design Notes for RF Pad Modules

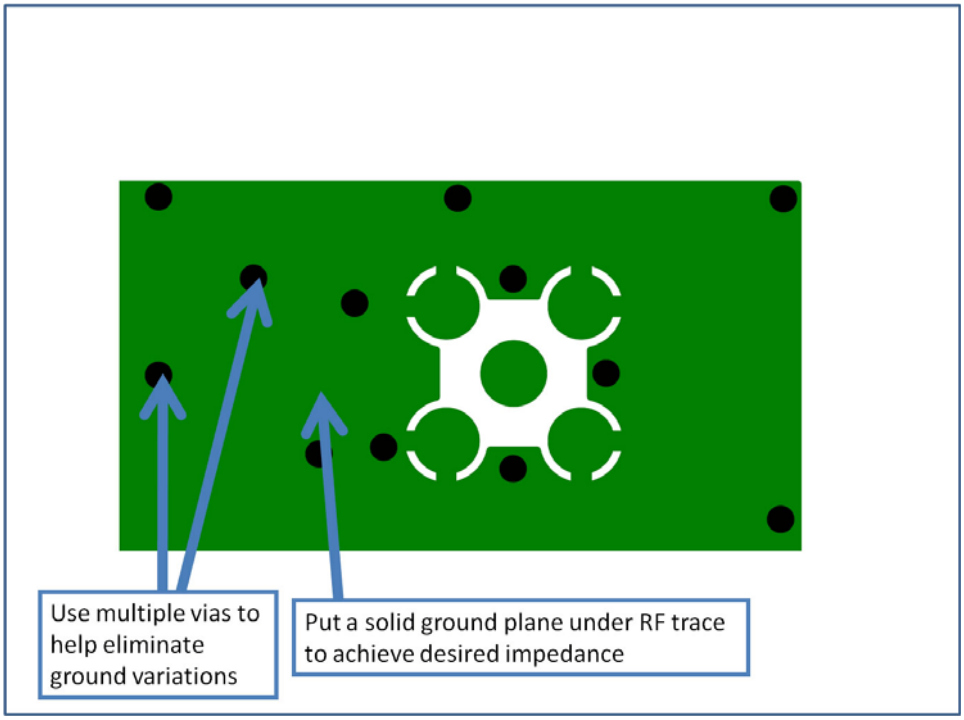
The RF Pad is a soldered antenna connection. The RF signal travels from pin 36 on the module to the antenna through an RF trace transmission line on the PCB. Please note that any additional components between the module and antenna will violate modular certification. The RF trace should have a controlled impedance of 50 ohms. We recommend using a microstrip trace, although coplanar waveguide may also be used if more isolation is needed. Microstrip generally requires less area on the PCB than coplanar waveguide. Stripline is not recommended because sending the signal to different PCB layers can introduce matching and performance problems.

It is essential to follow good design practices when implementing the RF trace on a PCB. The following figures show a layout example of a host PCB that connects an RF Pad module to a right angle, through hole RPSMA jack. The top two layers of the PCB have a controlled thickness dielectric material in between. The second layer has a ground plane which runs underneath the entire RF Pad area. This ground plane is a distance d, the thickness of the dielectric, below the top layer. The top layer has an RF trace running from pin 36 of the module to the RF pin of the RPSMA connector. The RF trace's width determines the impedance of the transmission line with relation to the ground plane. Many online tools can estimate this value, although the PCB manufacturer should be consulted for the exact width. Assuming d=0.025", and that the dielectric has a relative permittivity of 4.4, the width in this example will be approximately 0.045" for a 50 ohm trace. This trace width is a good fit with the module footprint's 0.060" pad width. Using a trace wider than the pad width is not recommended, and using a very narrow trace (under 0.010") can cause unwanted RF loss. The length of the trace is minimized by placing the RPSMA jack close to the module. All of the grounds on the jack and the module are connected to the ground planes directly or through closely placed vias. Any ground fill on the top layer should be spaced at least twice the distance d (in this case, at least 0.050") from the microstrip to minimize their interaction.

Implementing these design suggestions will help ensure that the RF Pad module performs to its specifications.



PCB Layer 1 of RF Layout Example



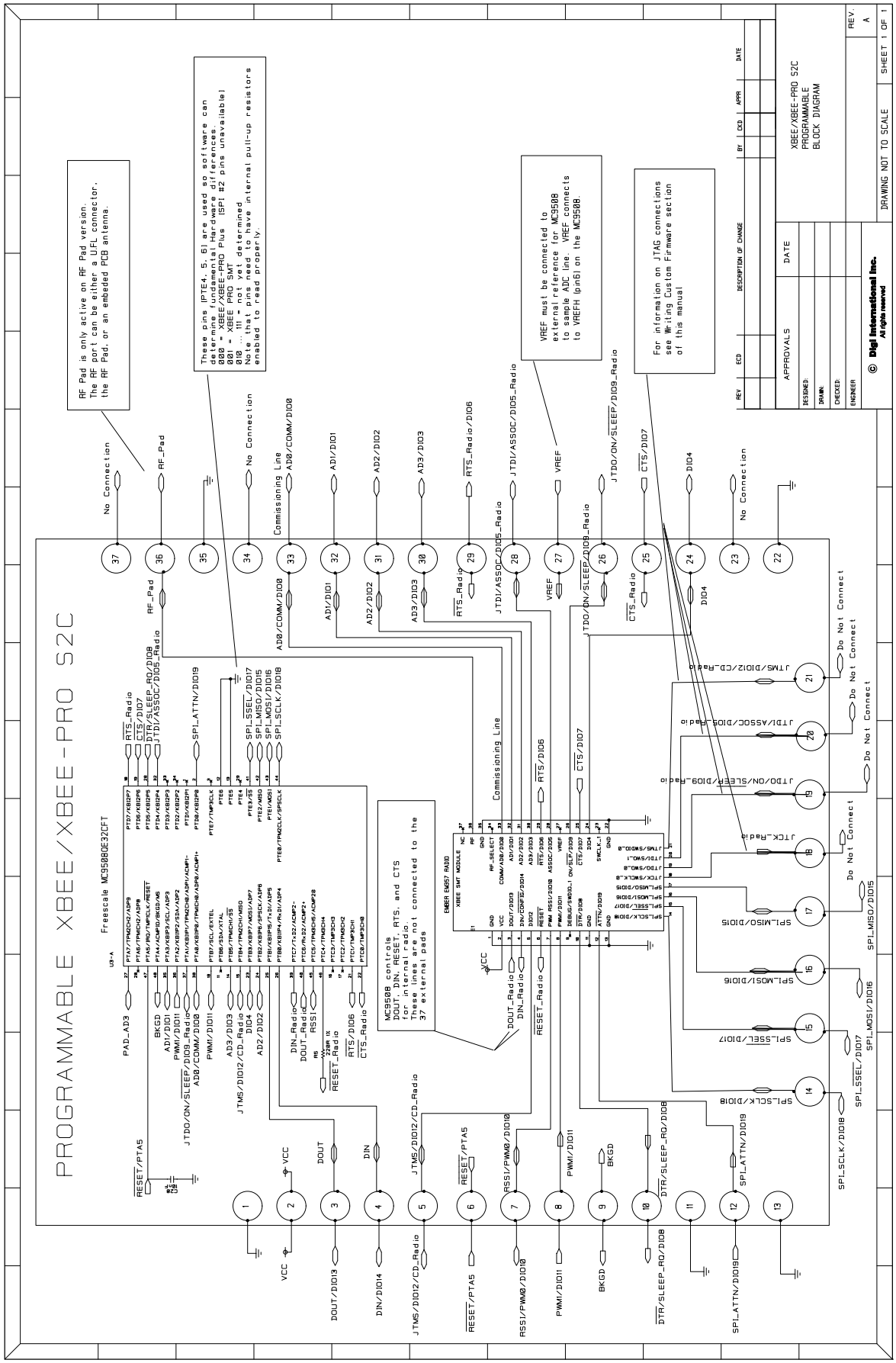
PCB Layer 2 of RF Layout Example

Module Operation for Programmable Variant

The modules with the programmable option have a secondary processor with 32k of flash and 2k of RAM. This allows module integrators to put custom code on the XBee module to fit their own unique needs. The DIN, DOUT, RTS, CTS, and RESET lines are intercepted by the secondary processor to allow it to be in control of the data transmitted and received. All other lines are in parallel and can be controlled by either the EM357 or the MC9SO8QE micro (see Block Diagram for details). The EM357 by default has control of certain lines. These lines can be released by the EM357 by sending the proper command(s) to disable the desired DIO line(s) (see XBee Command Reference Tables).

In order for the secondary processor to sample with ADCs, the XBee pin 27 (VREF) must be connected to a reference voltage.

Digi provides a bootloader that can take care of programming the processor over the air or through the serial interface. This means that over the air updates can be supported through an XMODEM protocol. The processor can also be programmed and debugged through a one wire interface BKGD (Pin 9).


PROGRAMMABLE XBEE/XBEE-PRO S2C
Freescale MC9S08QE32CFT
RF Pad is only active on RF Pad version.
The RF port can be either a U.FL connector,
the RF Pad, or an embeded PCB antenna.
These pins (PTE4, 5, 6) are used as software can
determine which hardware platform is used.
000 = XBEE/XBEE-PRO Plus (SPI #2 pins unavailable)
001 = XBEE PRO SMT
010 ... 111 = not yet determined
Note that pins need to have internal pull-up resistors
enabled to read properly.
VREF must be connected to
external reference for MC9S08
to sample ADC line. VREF connects
to VREFH (pin6) on the MC9S08.
For information on JTAG connections
see Writing Custom Firmware section
of this manual
MC9S08 controls
DOUT, DIN, RESET, RTS, and CTS
for an Xtena radio.
These lines are not connected to the
37 external pads
XBEE/XBEE-PRO S2C
PROGRAMMABLE
BLOCK DIAGRAM
APPROVALS
DRAWING NOT TO SCALE
SHEET 1 OF 1
© Digi International Inc.
All rights reserved

XBee Programmable Bootloader

Overview

The XBee Programmable module is equipped with a Freescale MC9S08QE32 application processor. This application processor comes with a supplied bootloader. This section describes how to interface the customer's application code running on this processor to the XBee Programmable module's supplied bootloader.

The first section discusses how to initiate firmware updates using the supplied bootloader for wired and over-the-air updates.

Bootloader Software Specifics

Memory Layout

Figure 1 shows the memory map for the MC9S08QE32 application processor.

The supplied bootloader occupies the bottom pages of the flash from 0xF200 to 0xFFFF. Application code cannot write to this space.

The application code can exist in Flash from address 0x8400 to 0xF1BC. 1k of Flash from 0x8000 to 0x83FF is reserved for Non Volatile Application Data that will not be erased by the bootloader during a flash update.

A portion of RAM is accessible by both the application and the bootloader. Specifically, there is a shared data region used by both the application and the bootloader that is located at RAM address 0x200 to 0x215. Application code should not write anything to BLResetCause or AppResetCause unless informing the bootloader of the impending reset reason. The Application code should not clear BLResetCause unless it is handling the unexpected reset reason.

To prevent a malfunctioning application from running forever, the Bootloader increments BLResetCause after each watchdog or illegal instruction reset. If this register reaches above 0x10 the bootloader will stop running the application for a few minutes to allow an OTA or Local update to occur. If no update is initiated within the time period, BLResetCause is cleared and the application is started again. To prevent unexpected halting of the application, the application shall clear or decrement BLResetCause just before a pending reset. To disable this feature, the application shall clear BLResetCause at the start of the application.

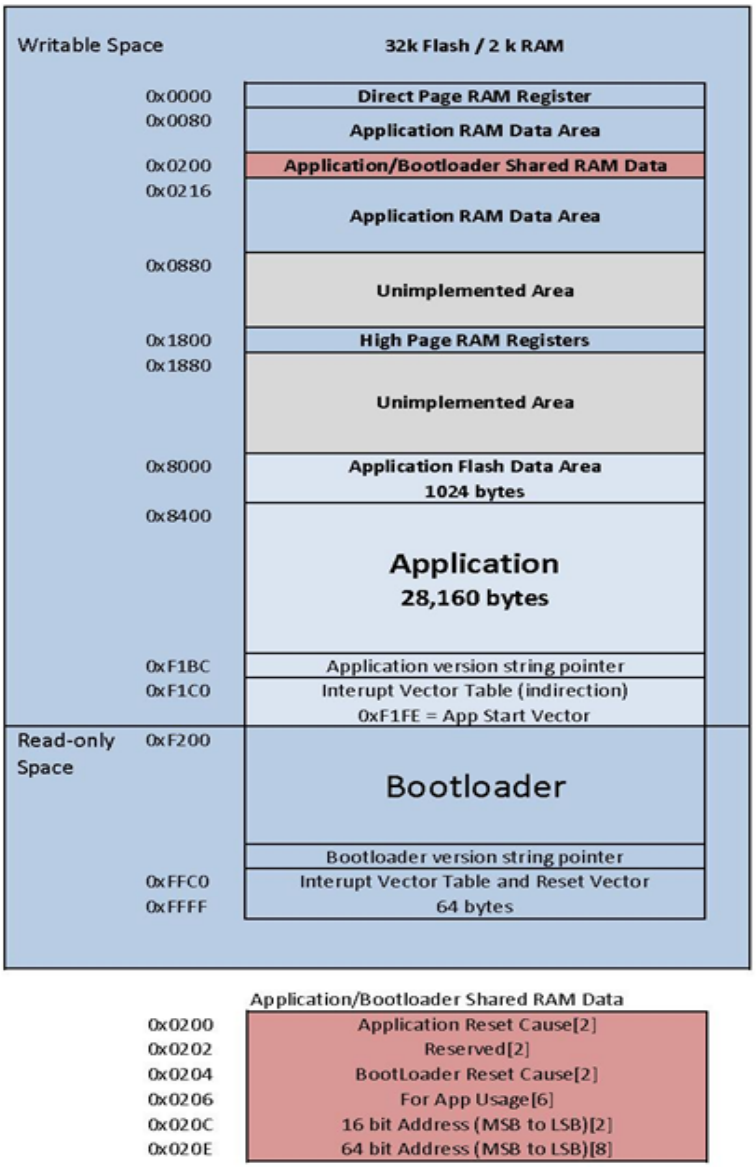


Figure 1: MC9S08QE32 Memory Map

Operation

Upon reset of any kind, the execution control begins with the bootloader.

If the reset cause is Power-On reset (POR), Pin reset (PIN), or Low Voltage Detect (LVD) reset (LVD) the bootloader will not jump to the application code if the override bits are set to RTS(D7)=1, DTR(D5)=0, and DIN(B0)=0. Otherwise, the bootloader writes the reset cause "NOTHING" to the shared data region, and jumps to the Application.

Reset causes are defined in the file *common. h* in an enumeration with the following definitions:

```
typedef enum {
  BL_CAUSE_NOTHING   = 0x0000,  //PIN, LVD, POR
  BL_CAUSE_NOTHING_COUNT   = 0x0001,//BL_Reset_Cause counter
// Bootloader increments cause every reset
  BL_CAUSE_BAD_APP   = 0x0010,//Bootloader considers APP invalid
} BL_RESET_CAUSES;
typedef enum {
   APP_CAUSE_NOTHING         = 0x0000,
   APP_CAUSE_USE001          = 0x0001,
// 0x0000 to 0x00FF are considered valid for APP use.
   APP_CAUSE_USE255          = 0x00FF,
   APP_CAUSE_FIRMWARE_UPDATE = 0x5981,
   APP_CAUSE_BYPASS_MODE     = 0x4682,
   APP_CAUSE_BOOTLOADER_MENU = 0x6A18,
} APP_RESET_CAUSES;
```

Otherwise, if the reset cause is a "watchdog" or other reset, the bootloader checks the shared memory region for the APP_RESET_CAUSE. If the reset cause is:

1. "APP_CAUSE_NOTHING" or 0x0000 to 0x00FF, the bootloader increments the BL_RESET_CAUSES, verifies that it is still less than BL_CAUSE_BAD_APP, and jumps back to the application. If the Application does not clear the BL_RESET_CAUSE, it can prevent an infinite loop of running a bad application that continues to perform illegal instructions or watchdog resets.
2. "APP_CAUSE_FIRMWARE_UPDATE", the bootloader has been instructed to update the application "over-the-air" from a specific 64-bit address. In this case, the bootloader will attempt to initiate an Xmodem transfer from the 64-bit address located in shared RAM.
3. "APP_CAUSE_BYPASS_MODE", the bootloader executes bypass mode. This mode passes the local UART data directly to the EM357 allowing for direct communication with the EM357. The only way to exit bypass mode is to reset or power cycle the module.

If none of the above is true, the bootloader will enter "Command mode". In this mode, users can initiate firmware downloads both wired and over-the-air, check application/bootloader version strings, and enter Bypass mode.

Application version string

Figure 1 shows an "Application version string pointer" area in application flash which holds the pointer to where the application version string resides. The application's linker command file ultimately determines where this string is placed in application flash.

It is preferable that the application version string be located at address 0x8400 for MC9S08QE32 parts. The application string can be any characters terminated by the NULL character (0x00). There is not a strict limit on the number of characters in the string, but for practical purposes should be kept under 100 bytes including the terminating NULL character. During an update the bootloader erases the entire application from 0x8400 on. The last page has the vector table specifically the redirected reset vector. The version string pointer and reset vector are used to determine if the application is valid.

Application Interrupt Vector table and Linker Command File

Since the bootloader flash region is read-only, the interrupt vector table is redirected to the region 0xF1C0 to 0xF1FD so that application developers can use hardware interrupts. Note that in order for Application interrupts to function properly, the Application's linker command file (*.prm extension) must be modified appropriately to allow the linker to place the developers code in the correct place in memory. For example, the developer desires to use the serial communications port SCI1 receive interrupt. The developer would add the following line to the Codewarrior linker command file for the project:

VECTOR ADDRESS 0x0000F1E0 vSci1Rx

This will inform the linker that the interrupt function "vSci1Rx()" should be placed at address 0x0000F1E0. Next, the developer should add a file to their project "vector_table.c" that creates an array of function pointers to the ISR routines used by the application.

```
extern void _Startup(void);/* _Startup located in Start08.c */
extern void vSci1Rx(void);/* sci1 rx isr */
extern short iWriteToSci1(unsigned char *);
void vDummyIsr(void);
#pragma CONST_SEG VECTORS
void (* const vector_table[])(void) = /* Relocated Interrupt vector table */{
    vDummyIsr,/* Int.no. 0 Vtpm3ovf (at F1C0)Unassigned */
    vDummyIsr, /* Int.no. 1 Vtpm3ch5 (at F1C2) Unassigned */
    vDummyIsr, /* Int.no. 2 Vtpm3ch4 (at F1C4) Unassigned */
    vDummyIsr, /* Int.no. 3 Vtpm3ch3 (at F1C6) Unassigned */
    vDummyIsr, /* Int.no. 4 Vtpm3ch2 (at F1C8) Unassigned */
    vDummyIsr, /* Int.no. 5 Vtpm3ch1 (at F1CA) Unassigned */
    vDummyIsr, /* Int.no. 6 Vtpm3ch0 (at F1CC) Unassigned */
    vDummyIsr, /* Int.no. 7 Vrtc (at F1CE) Unassigned */
    vDummyIsr, /* Int.no. 8 Vsci2tx (at F1D0) Unassigned */
    vDummyIsr, /* Int.no. 9 Vsci2rx (at F1D2) Unassigned */
    vDummyIsr, /* Int.no. 10 Vsci2err (at F1D4) Unassigned */
    vDummyIsr, /* Int.no. 11 Vacmpx (at F1D6) Unassigned */
    vDummyIsr, /* Int.no. 12 Vadc (at F1D8) Unassigned */
    vDummyIsr, /* Int.no. 13 Vkeyboard (at F1DA) Unassigned */
    vDummyIsr, /* Int.no. 14 Viic (at F1DC) Unassigned */
    vDummyIsr, /* Int.no. 15 Vsci1tx (at F1DE) Unassigned */
    vSci1Rx, /* Int.no. 16 Vsci1rx (at F1E0) SCI1RX */
    vDummyIsr, /* Int.no. 17 Vsci1err (at F1E2) Unassigned */
    vDummyIsr, /* Int.no. 18 Vspi (at F1E4) Unassigned */
    vDummyIsr, /* Int.no. 19 VReserved12 (at F1E6) Unassigned */
    vDummyIsr, /* Int.no. 20 Vtpm2ovf (at F1E8) Unassigned */
    vDummyIsr, /* Int.no. 21 Vtpm2ch2 (at F1EA) Unassigned */
    vDummyIsr, /* Int.no. 22 Vtpm2ch1 (at F1EC) Unassigned */
    vDummyIsr, /* Int.no. 23 Vtpm2ch0 (at F1EE) Unassigned */
    vDummyIsr, /* Int.no. 24 Vtpm1ovf (at F1F0) Unassigned */
    vDummyIsr, /* Int.no. 25 Vtpm1ch2 (at F1F2) Unassigned */
    vDummyIsr, /* Int.no. 26 Vtpm1ch1 (at F1F4) Unassigned */
    vDummyIsr, /* Int.no. 27 Vtpm1ch0 (at F1F6) Unassigned */
```

```
                    vDummyIsr,  /* Int.no. 28 Vlvd (at F1F8)      Unassigned */
                    vDummyIsr,  /* Int.no. 29 Virq (at F1FA)      Unassigned */
                    vDummyIsr,  /* Int.no. 30 Vswi (at F1FC)      Unassigned */
                    _Startup    /* Int.no. 31 Vreset (at F1FE)    Reset vector */
};
void vDummyIsr(void){
 for(;;){
                  if(iWriteToSci1("STUCK IN UNASSIGNED ISR\n\r>"));
                 }
}
```

The interrupt routines themselves can be defined in separate files. The "vDummyIsr" function is used in conjunction with "iWritetoSci1" for debugging purposes.

Bootloader Menu Commands

The bootloader accepts commands from both the local UART and OTA. All OTA commands sent must be Unicast with only 1 byte in the payload for each command. A response will be returned to the sender. All Broadcast and multiple byte OTA packets are dropped to help prevent general OTA traffic from being interpreted as a command to the bootloader while in the menu.

Bypass Mode - "B"

The bootloader provides a "bypass" mode of operation that essentially connects the SCI1 serial communications peripheral of the freescale mcu to the EM357's serial Uart channel. This allows direct communication to the EM357 radio for the purpose of firmware and radio configuration changes. Once in bypass mode, the X-CTU utility can change modem configuration and/or update EM357 firmware. Bypass mode automatically handles any baud rate up to 115.2kbps. Note that this command is unavailable when module is accessed remotely.

Update Firmware - "F"

The "F" command initiates a firmware download for both wired and over-the-air configurations. Depending on the source of the command (received via Over the Air or local UART), the download will proceed via wired or over-the-air respectively.

Adjust Timeout for Update Firmware - "T"

The "T" command changes the timeout before sending a NAK by Base-Time*2^(T). The Base-Time for the local UART is different than the Base-Time for Over the Air. During a firmware update, the bootloader will automatically increase the Timeout if repeat packets are received or multiple NAKs for the same packet without success occur.

Application Version String - "A"

The "A" command provides the version of the currently loaded application. If no application is present, "Unkown" will be returned.

Bootloader Version String - "V"

The "V" command provides the version of the currently loaded bootloader. The version will return a string in the format BLFFF-HHH-XYZ_DDD where FFF represents the Flash size in kilo bytes, HHH is the hardware, XYZ is the version, and DDD is the preferred XMODEM packet size for updates. Double the preferred packet size is also possible, but not guaranteed. For example "BL032-2B0-023_064" will take 64 byte CRC XMODEM payloads and may take 128 byte CRC XMODEM payloads also. In this case, both 64 and 128 payloads are handled, but the 64 byte payload is preferred for better Over the Air reliability.

Bootloader Version BL032-2x0-025_064 only operates at 9600 baud on the local UART as well as communications to the EM357 Radio. A newer version of the Bootloader BL032-2x0-033_064 or newer BL032-2B0-XXX_064 has changed the baud rate to 115200 between the Programmable and the EM357

Radio. The EM357 is also set to 115200 as the default baud rate. The default rate of the programmable local UART is also set to 115200, however, the local UART has an auto baud feature added to detect if the UART is at the wrong baud rate. If a single character is sent, it will automatically switch to 115200 or 9600 baud.

Firmware Updates

Wired Updates

A user can update their application using the bootloader in a wired configuration with the following steps:

a. Plug XBee programmable module into a suitable serial port on a PC.

b. Open a hyperterminal (or similar dumb terminal application) session with 115200 baud, no parity, and 8 data bits with one stop bit.

c. Hit Enter to display the bootloader menu.

d. Hit the "F" key to initiate a wired firmware update.

e. A series of "C" characters Will be displayed within the hyperterminal window. At this point, select the "transfer->send file" menu item. Select the desired flat binary output file.

f. Select "Xmodem" as the protocol.

g. Click "Send" on the "Send File" dialog. The file will be downloaded to the XBee Programmable module. Upon a successful update, the bootloader will jump to the newly loaded application.

Over-The-Air updates

A user can update their application using the bootloader in an "over-the-air" configuration with the following steps…(This procedure assumes that the bootloader is running and not the application. The EM357 baud rate of the programmable module must be set to 115200 baud. The

bootloader only operates at 115200 baud between the Radio and programmable bootloader. The application must be programmed with some way to support returning to the bootloader in order to support Over the Air (OTA) updates without local intervention.)

a. The XBee module sending the file OTA (Host module) should be set up with a series 2 Xbee module with transparent mode firmware.

b. The XBee Programmable module receiving the update (remote module) is configured with API firmware.

c. Open a hyperterminal session to the host module with no parity, no hardwareflow control, 8 data bits and 1 stop bit. (The host module does not have to operate at the same baud rate as the remote module.) For faster updates and less latency due to the UART, set the host module to a faster baud rate. (i.e. 115200)

d.Enter 3 pluses "+++" to place the EM357 in command mode. (or XCTU's "Modem Configuration" tab can be used to set the correct parameters)

e. Set the Host Module destination address to the target module's 64 bit address that the host module will update (ATDH aabbccdd, ATDL eeffgghh, ATCN, where aabbccddeeffgghh is the hexadecimal 64 bit address of the target module).

f. Hit Enter and the bootloader command menu will be displayed from the remote module. (Note that the option "B" doesn't exist for OTA)

g. Hit the "F" key to cause the remote module to request the new firmware file over-the-air.

h. The host module will begin receiving "C" characters indicating that the remote module is requesting an Xmodem CRC transfer. Using XCTU or another terminal program, Select "XMODEM" file transfer. Select the Binary file to upload/transfer. Click Send to start the transfer. At the conclusion of a successful transfer, the bootloader will jump to the newly loaded application.

Output File Configuration

BKGD Programming

P&E Micro provides a background debug tool that allows flashing applications on the MC9S08QE parts through their background debug mode port. By default, the Codewarrior tool produces an "ABS" output file for use in programming parts through the background debug interface. The programmable XBee from the factory has the BKGD debugging capability disabled. In order to debug, a bootloader with the debug interface enabled needs to be loaded on the secondary processor or a stand-alone app needs to be loaded.

Bootloader updates

The supplied bootloader requires files in a "flat binary" format which differs from the default ABS file produced. The Codewarrior tool also produces a S19 output file. In order to successfully flash new applications, the S19 file must be converted into the flat binary format. Utilities are available on the web that will convert S19 output to "BIN" outputs. Often times, the "BIN" file conversion will pad the addresses from 0x0000 to the code space with the same number. (Often 0x00 or 0xFF) These extra bytes before the APP code starts will need to be deleted from the bin file before the file can be transferred to the bootloader.

2. RF Module Operation

Serial Communications

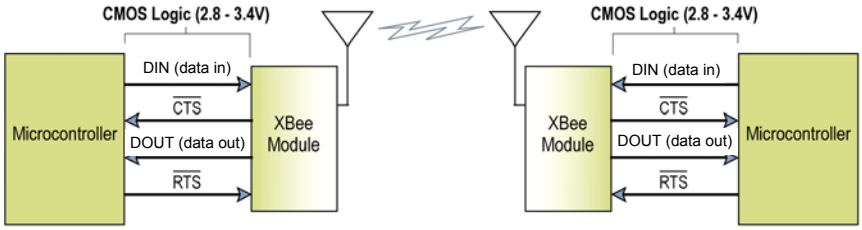
XBee RF Modules interface to a host device through a serial port. Through its serial port, the module can communicate with any logic and voltage compatible UART, through a level translator to any serial device (for example, through a RS-232 or USB interface board), or through a Serial Peripheral Interface, which is a synchronous interface to be described later.

Two Wire serial Interface (TWI) is also available, but not supported by Digi. For information on the TWI, see the EM357 specification.

UART Data Flow

Devices that have a UART interface can connect directly to the pins of the RF module as shown in the figure below.

System Data Flow Diagram in a UART-interfaced environment
(Low-asserted signals distinguished with horizontal line over signal name.)

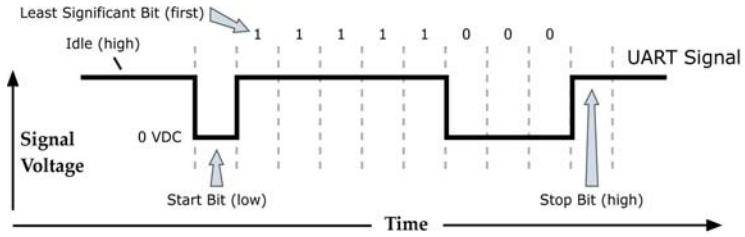


Serial Data

Data enters the module UART through the DIN (pin 4) as an asynchronous serial signal. The signal should idle high when no data is being transmitted.

Each data byte consists of a start bit (low), 8 data bits (least significant bit first) and a stop bit (high). The following figure illustrates the serial bit pattern of data passing through the module.

UART data packet 0x1F (decimal number "31") as transmitted through the RF module
Example Data Format is 8-N-1 (bits - parity - # of stop bits)



Serial communications depend on the two UARTs (the microcontroller's and the RF module's) to be configured with compatible settings (baud rate, parity, start bits, stop bits, data bits).

The UART baud rate, parity, and stop bits settings on the XBee module can be configured with the BD, NB, and SB commands respectively. See the command table in chapter 10 for details.

SPI Communications

The XBee modules support SPI communications in slave mode. Slave mode receives the clock signal and data from the master and returns data to the master. The SPI port uses the following signals on the XBee:

- SPI_MOSI (Master Out, Slave In) - inputs serial data from the master
- SPI_MISO (Master In, Slave Out) - outputs serial data to the master
- SPI_SCLK (Serial Clock) - clocks data transfers on MOSI and MISO

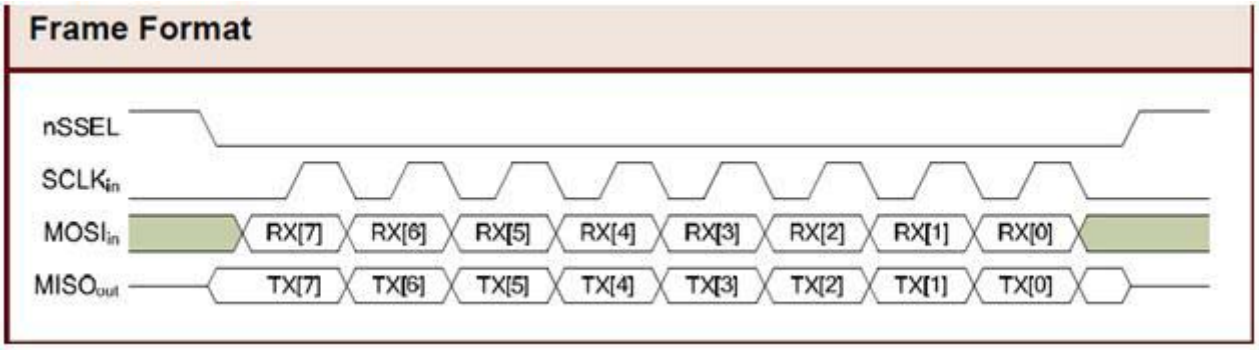
- SPI_SSEL (Slave Select) - enables serial communication with the slave

The above four pins are standard for SPI. This module also supports an additional pin, which may be configured to alert the SPI master when it has data to send. This pin is called SPI_ATTN. If the master monitors this pin (through polling or interrupts), it can know when it needs to receive data from the module. SPI_ATTN asserts whenever it has data to send and it remains asserted until all available data has been shifted out to the SPI master.

In this mode, the following apply:

- Data/Clock rates of up to 5 Mbps are possible
- Data is MSB first
- Frame Format mode 0 is used (see below)

Frame Format for SPI Communications



SPI Operation

When the slave select (SPI_SSEL) signal is asserted by the master, SPI transmit data is driven to the output pin (SPI_MISO), and SPI data is received from the input pin SPI_MOSI. The SPI_SSEL pin has to be asserted to enable the transmit serializer to drive data to the output signal SPI_MISO. A falling edge on SPI_SSEL resets the SPI slave shift registers.

If the SPI_SCLK is present, the SPI_MISO line is always driven whether with or without the SPI_SSEL line driven. This is a known issue with the Ember EM357 chip, and makes additional hardware necessary if multiple slaves are using the same bus as the XBee.

If the input buffer is empty, the SPI serializer transmits a busy token (0xFF). Otherwise, all transactions on the SPI port use API operation. See Chapter 9 - API Operation for more information.

The SPI slave controller must guarantee that there is time to move new transmit data from the transmit buffer into the hardware serializer. To provide sufficient time, the SPI slave controller inserts a byte of padding at the start of every new string of transmit data. Whenever the transmit buffer is empty and data is placed into the transmit buffer, the SPI hardware inserts a byte of padding onto the front of the transmission as if this byte were placed there by software.

Serial Port Selection

In the default configuration the UART and SPI ports will both be configured for serial port operation.

If both interfaces are configured, serial data will go out the UART until the SPI_SSEL signal is asserted. Thereafter, all serial communications will operate on the SPI interface.

If only the UART is enabled, then only the UART will be used, and SPI_SSEL will be ignored. If only the SPI is enabled, then only the SPI will be used.

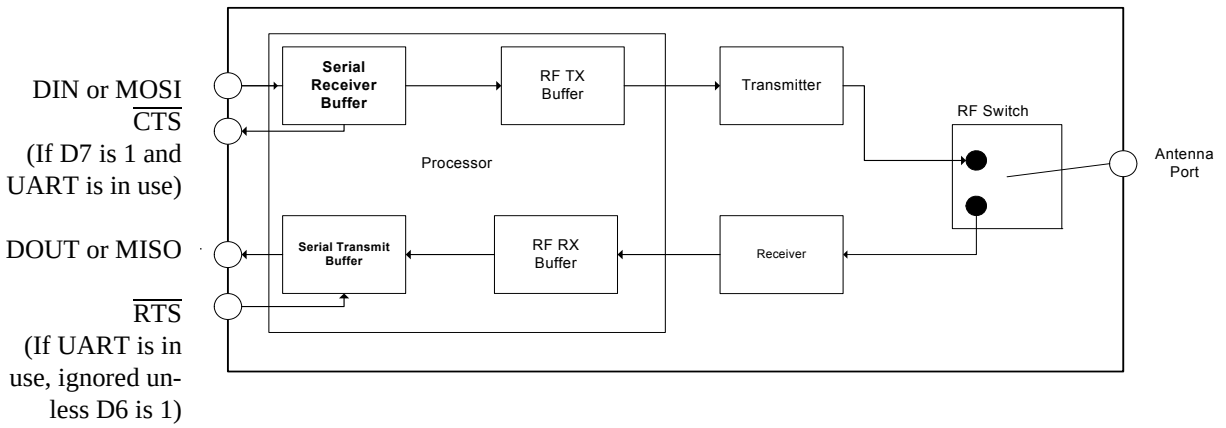
If neither serial port is enabled, the module will not support serial operations and all communications must occur over the air. All data that would normally go to the serial port is discarded.

Serial Buffers

The XBee modules maintain small buffers to collect received serial and RF data, which is illustrated in the figure below. The serial receive buffer collects incoming serial characters and holds them until they can be processed.

The serial transmit buffer collects data that is received via the RF link that will be transmitted out the UART or SPI port.

Internal Data Flow Diagram



Serial Receive Buffer

When serial data enters the RF module through the serial port, the data is stored in the serial receive buffer until it can be processed. Under certain conditions, the module may receive data when the serial receive buffer is already full. In that case the data is discarded.

The serial receive buffer becomes full when data is streaming into the serial port faster than it can be processed and sent over the air (OTA). While the speed of receiving the data on the serial port can be much faster than the speed of transmitting to data for a short period, sustained operation in that mode will cause data to be dropped due to running out of places in the module to put the data. Some things that may delay over the air transmissions are address discovery, route discovery, and retransmissions. Processing received RF data can also take away time and resources for processing incoming serial data.

If the UART is the serial port and CTS flow control is enabled, the external data source is alerted when the receive buffer is almost full. Then the host holds off sending data to the module until the module asserts CTS again, allowing more data to come in.

If the SPI is the serial port, no hardware flow control is available. It is the user's responsibility to ensure that that receive buffer is not overflowed. One reliable strategy is to wait for a TX_STATUS response after each frame sent to ensure that the module has had time to process it.

Serial Transmit Buffer

When RF data is received, the data is moved into the serial transmit buffer and sent out the UART or SPI port. If the serial transmit buffer becomes full enough such that all data in a received RF packet won't fit in the serial transmit buffer, the entire RF data packet is dropped.

Cases in which the serial transmit buffer may become full resulting in dropped RF packets:

1. If the RF data rate is set higher than the interface data rate of the module, the module could receive data faster than it can send the data to the host.
2. If the host does not allow the module to transmit data out from the serial transmit buffer because of being held off by hardware flow control.

UART Flow Control

The $\overline{\text{RTS}}$ and $\overline{\text{CTS}}$ module pins can be used to provide RTS and/or CTS flow control. CTS flow control provides an indication to the host to stop sending serial data to the module. RTS flow control allows the host to signal the module to not send data in the serial transmit buffer out the UART. RTS and CTS flow control are enabled using the D6 and D7 commands. Please note that serial port flow control is not possible when using the SPI port.

CTS Flow Control

If CTS flow control is enabled (D7 command), when the serial receive buffer is 17 bytes away from being full, the module de-asserts $\overline{\text{CTS}}$ (sets it high) to signal to the host device to stop sending serial data. $\overline{\text{CTS}}$ is re-asserted after the serial receive buffer has 34 bytes of space.

RTS Flow Control

If RTS flow control is enabled (D6 command), data in the serial transmit buffer will not be sent out the DOUT pin as long as $\overline{\text{RTS}}$ is de-asserted (set high). The host device should not de-assert $\overline{\text{RTS}}$ for long periods of time to avoid filling the serial transmit buffer. If an RF data packet is received, and the serial transmit buffer does not have enough space for all of the data bytes, the entire RF data packet will be discarded.

Note: If the XBee is sending data out the UART when $\overline{\text{RTS}}$ is de-asserted (set high), the XBee could send up to 5 characters out the UART or SPI port after $\overline{\text{RTS}}$ is de-asserted.

Break Control

If break is enabled for over five seconds, the XBee will reset. Then it will boot with default baud settings into command mode.

This break function will be disabled if either P3 or P4 are not enabled.

Serial Interface Protocols

The XBee modules support both transparent and API (Application Programming Interface) serial interfaces.

Transparent Operation

When operating in transparent mode, the modules act as a serial line replacement. All UART or SPI data received through the DIN or MOSI pin is queued up for RF transmission. When RF data is received, the data is sent out through the serial port. The module configuration parameters are configured using the AT command mode interface. Please note that transparent operation is not provided when using the SPI.

Data is buffered in the serial receive buffer until one of the following causes the data to be packetized and transmitted:

- No serial characters are received for the amount of time determined by the RO (Packetization Timeout) parameter. If RO = 0, packetization begins when a character is received.
- The Command Mode Sequence (GT + CC + GT) is received. Any character buffered in the serial receive buffer before the sequence is transmitted.
- The maximum number of characters that will fit in an RF packet is received.

API Operation

API operation is an alternative to transparent operation. The frame-based API extends the level to which a host application can interact with the networking capabilities of the module. When in API mode, all data entering and leaving the module is contained in frames that define operations or events within the module.

Transmit Data Frames (received through the serial port) include:

- RF Transmit Data Frame
- Command Frame (equivalent to AT commands)

Receive Data Frames (sent out the serial port) include:

- RF-received data frame
- Command response
- Event notifications such as reset, associate, disassociate, etc.

The API provides alternative means of configuring modules and routing data at the host application layer. A host application can send data frames to the module that contain address and payload information instead of using command mode to modify addresses. The module will send data frames to the application containing status packets; as well as source, and payload information from received data packets.

The API operation option facilitates many operations such as the examples cited below:

-> Transmitting data to multiple destinations without entering Command Mode

-> Receive success/failure status of each transmitted RF packet

-> Identify the source address of each received packet

A Comparison of Transparent and API Operation

The following table compares the advantages of transparent and API modes of operation:

Transparent Operation Features	
Simple Interface	All received serial data is transmitted unless the module is in command mode.
Easy to support	It is easier for an application to support transparent operation and command mode
API Operation Features	
Easy to manage data transmissions to multiple destinations	Transmitting RF data to multiple remotes only requires changing the address in the API frame. This process is much faster than in transparent operation where the application must enter AT command mode, change the address, exit command mode, and then transmit data. Each API transmission can return a transmit status frame indicating the success or reason for failure.
Received data frames indicate the sender's address	All received RF data API frames indicate the source address.
Advanced ZigBee addressing support	API transmit and receive frames can expose ZigBee addressing fields including source and destination endpoints, cluster ID and profile ID. This makes it easy to support ZDO commands and public profile traffic.
Advanced networking diagnostics	API frames can provide indication of IO samples from remote devices, and node identification messages.
Remote Configuration	Set / read configuration commands can be sent to remote devices to configure them as needed using the API.

As a general rule of thumb, API mode is recommended when a device:

- sends RF data to multiple destinations
- sends remote configuration commands to manage devices in the network
- receives RF data packets from multiple devices, and the application needs to know which device sent which packet
- must support multiple ZigBee endpoints, cluster IDs, and/or profile IDs
- uses the ZigBee Device Profile services.

API mode is required when:

- using Smart Energy firmware
- using SPI for the serial port
- receiving I/O samples from remote devices
- using source routing

If the above conditions do not apply (e.g. a sensor node, router, or a simple application), then transparent operation might be suitable. It is acceptable to use a mixture of devices running API mode and transparent mode in a network.

Modes of Operation

Idle Mode

When not receiving or transmitting data, the RF module is in Idle Mode. The module shifts into the other modes of operation under the following conditions:

- Transmit Mode (Serial data in the serial receive buffer is ready to be packetized)
- Receive Mode (Valid RF data is received through the antenna)
- Sleep Mode (End Devices only)
- Command Mode (Command Mode Sequence is issued, not available with Smart Energy software or when using the SPI port)

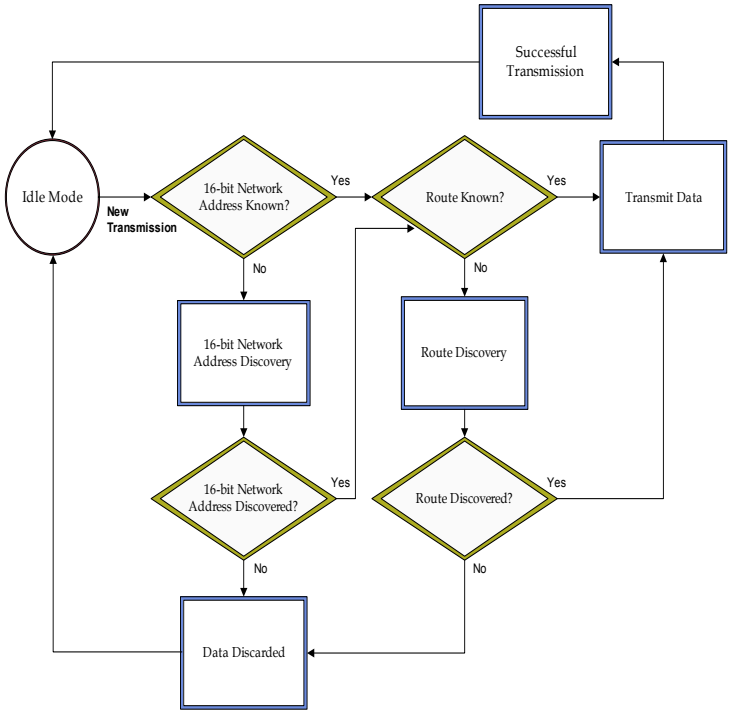
Transmit Mode

When serial data is received and is ready for packetization, the RF module will exit Idle Mode and attempt to transmit the data. The destination address determines which node(s) will receive the data.

Prior to transmitting the data, the module ensures that a 16-bit network address and route to the destination node have been established.

If the destination 16-bit network address is not known, network address discovery will take place. If a route is not known, route discovery will take place for the purpose of establishing a route to the destination node. If a module with a matching network address is not discovered, the packet is discarded. The data will be transmitted once a route is established. If route discovery fails to establish a route, the packet will be discarded.

Transmit Mode Sequence



When data is transmitted from one node to another, a network-level acknowledgement is transmitted back across the established route to the source node. This acknowledgement packet indicates to the source node that the data packet was received by the destination node. If a network acknowledgement is not received, the source node will re-transmit the data.

It is possible in rare circumstances for the destination to receive a data packet, but for the source to not receive the network acknowledgment. In this case, the source will retransmit the data, which could cause the destination to receive the same data packet multiple times. The XBee modules do not filter out duplicate packets. The application should include provisions to address this potential issue

See Data Transmission and Routing in chapter 4 for more information.

Receive Mode

If a valid RF packet is received, the data is transferred to the serial transmit buffer.

Command Mode

To modify or read RF Module parameters, the module must first enter into Command Mode - a state in which incoming serial characters are interpreted as commands. Command Mode is only available over the UART when not using the Smart Energy firmware. The API Mode section in Chapter 9 describes an alternate means for configuring modules which is available with the SPI and with Smart Energy, as well as over the UART with ZB code.

AT Command Mode

To Enter AT Command Mode:

Send the 3-character command sequence "+++" and observe guard times before and after the command characters. [Refer to the "Default AT Command Mode Sequence" below.]

Default AT Command Mode Sequence (for transition to Command Mode):

- No characters sent for one second [GT (Guard Times) parameter = 0x3E8]
- Input three plus characters ("+++") within one second [CC (Command Sequence Character) parameter = 0x2B.]
- No characters sent for one second [GT (Guard Times) parameter = 0x3E8]

Once the AT command mode sequence has been issued, the module sends an "OK\r" out the UART pad. The "OK\r" characters can be delayed if the module has not finished transmitting received serial data.

When command mode has been entered, the command mode timer is started (CT command), and the module is able to receive AT commands on the UART port.

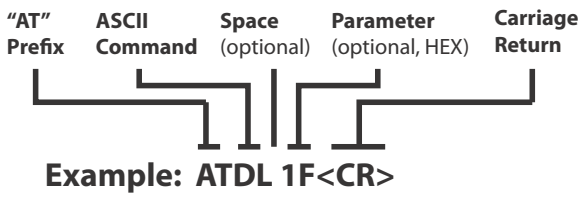
All of the parameter values in the sequence can be modified to reflect user preferences.

> NOTE: Failure to enter AT Command Mode is most commonly due to baud rate mismatch. By default, the BD (Baud Rate) parameter = 3 (9600 bps).

To Send AT Commands:

Send AT commands and parameters using the syntax shown below.

Syntax for sending AT Commands



> To read a parameter value stored in the RF module's register, omit the parameter field.

The preceding example would change the RF module Destination Address (Low) to "0x1F". To store the new value to non-volatile (long term) memory, subsequently send the WR (Write) command.

For modified parameter values to persist in the module's registry after a reset, changes must be saved to non-volatile memory using the WR (Write) Command. Otherwise, parameters are restored to previously saved values after the module is reset.

Command Response

When a command is sent to the module, the module will parse and execute the command. Upon successful execution of a command, the module returns an "OK" message. If execution of a command results in an error, the module returns an "ERROR" message.

Applying Command Changes

Any changes made to the configuration command registers through AT commands will not take effect until the changes are applied. For example, sending the BD command to change the baud rate will not change the actual baud rate until changes are applied. Changes can be applied in one of the following ways:

- The AC (Apply Changes) command is issued.
- AT command mode is exited.

To Exit AT Command Mode:

1. Send the ATCN (Exit Command Mode) command (followed by a carriage return).

   [OR]

2. If no valid AT Commands are received within the time specified by CT (Command Mode Timeout) Command, the RF module automatically returns to Idle Mode.

For an example of programming the RF module using AT Commands and descriptions of each configurable parameter, please see the Command Reference Table chapter.

Sleep Mode

Sleep modes allow the RF module to enter states of low power consumption when not in use. XBee RF modules support both pin sleep (sleep mode entered on pin transition) and cyclic sleep (module sleeps for a fixed time). XBee sleep modes are discussed in detail in chapter 7.

3. XBee ZigBee Networks

Introduction to ZigBee

ZigBee is an open global standard built on the IEEE 802.15.4 MAC/PHY. ZigBee defines a network layer above the 802.15.4 layers to support advanced mesh routing capabilities. The ZigBee specification is developed by a growing consortium of companies that make up the ZigBee Alliance. The Alliance is made up of over 300 members, including semiconductor, module, stack, and software developers.

ZigBee Stack Layers

The ZigBee stack consists of several layers including the PHY, MAC, Network, Application Support Sublayer (APS), and ZigBee Device Objects (ZDO) layers. Technically, an Application Framework (AF) layer also exists, but will be grouped with the APS layer in remaining discussions. The ZigBee layers are shown in the figure below.

A description of each layer appears in the following table:

ZigBee Layer	Description
PHY	Defines the physical operation of the ZigBee device including receive sensitivity, channel rejection, output power, number of channels, chip modulation, and transmission rate specifications. Most ZigBee applications operate on the 2.4 GHz ISM band at a 250kbps data rate. See the IEEE 802.15.4 specification for details.
MAC	Manages RF data transactions between neighboring devices (point to point). The MAC includes services such as transmission retry and acknowledgment management, and collision avoidance techniques (CSMA-CA).
Network	Adds routing capabilities that allows RF data packets to traverse multiple devices (multiple "hops") to route data from source to destination (peer to peer).
APS (AF)	Application layer that defines various addressing objects including profiles, clusters, and endpoints.
ZDO	Application layer that provides device and service discovery features and advanced network management capabilities.

Networking Concepts

Device Types

ZigBee defines three different device types: coordinator, router, and end device.

Node Types / Sample of a Basic ZigBee Network Topology

A **coordinator** has the following characteristics: It ...

- Selects a channel and PAN ID (both 64-bit and 16-bit) to start the network
- Can allow routers and end devices to join the network
- Can assist in routing data
- Cannot sleep--should be mains powered
- Can buffer RF data packets for sleeping end device children.

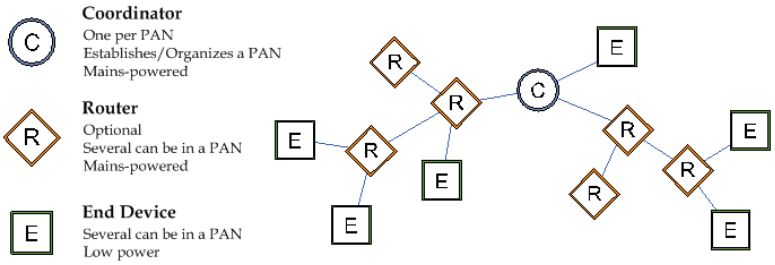
A **router** has the following characteristics: It ...

- •Must join a ZigBee PAN before it can transmit, receive, or route data
- •After joining, can allow routers and end devices to join the network
- •After joining, can assist in routing data
- •Cannot sleep--should be mains powered.
- •Can buffer RF data packets for sleeping end device children.

An **end device** has the following characteristics: It ...

- •Must join a ZigBee PAN before it can transmit or receive data
- •Cannot allow devices to join the network
- •Must always transmit and receive RF data through its parent, and cannot route data.
- •Can enter low power modes to conserve power and can be battery-powered.

An example of such a network is shown below:



In ZigBee networks, the coordinator must select a PAN ID (64-bit and 16-bit) and channel to start a network. After that, it behaves essentially like a router. The coordinator and routers can allow other devices to join the network and can route data.

After an end device joins a router or coordinator, it must be able to transmit or receive RF data through that router or coordinator. The router or coordinator that allowed an end device to join becomes the "parent" of the end device. Since the end device can sleep, the parent must be able to buffer or retain incoming data packets destined for the end device until the end device is able to wake and receive the data.

A module can only operate as one of the three device types. The device type is selected by configuration rather than by firmware image as was the case on earlier hardware platforms.

By default, the module operates as a router in transparent mode. To select coordinator operation, set CE to 1. To select end device operation, set SM to a non-zero value. To select router operation, both CE and SM must be 0.

One complication is that if a device is a coordinator and it needs to be changed into an end device, CE must be set back to 0 first. If not, the SM configuration will conflict with the CE configuration. Likewise, to change an end device into a coordinator, it must be changed into a router first.

Another complication is that default parameters for a router build don't always work very well for a coordinator build. For example:

DH/DL is 0 by default, which allows routers and end devices to send data to the coordinator when they first come up. If DH/DL is not changed from the default value when the device is changed to a coordinator, then the device will send data to itself, causing characters to be echoed back to the screen as they are typed. Since this is probably not the desired operation, DH/DL should be set to the broadcast address or some specific unicast address when the device is changed to a coordinator.

Another example is EO for smart energy builds. This value should be 08 for routers and end devices and it should be 02 for the coordinator to designate it as the trust center. Therefore, if using authentication, which is the normal case for Smart Energy builds, EO should be changed from 02 to 08 when CE is set to 1.

In general, when changing device types, it is the user's responsibility to ensure that parameters are set to be compatible with the new device type.

PAN ID

ZigBee networks are called personal area networks or PANs. Each network is defined with a unique PAN identifier (PAN ID). This identifier is common among all devices of the same network. ZigBee devices are either preconfigured with a PAN ID to join, or they can discover nearby networks and select a PAN ID to join.

ZigBee supports both a 64-bit and a 16-bit PAN ID. Both PAN IDs are used to uniquely identify a network. Devices on the same ZigBee network must share the same 64-bit and 16-bit PAN IDs. If multiple ZigBee networks are operating within range of each other, each should have unique PAN IDs.

The 16-bit PAN ID is used as a MAC layer addressing field in all RF data transmissions between devices in a network. However, due to the limited addressing space of the 16-bit PAN ID (65,535 possibilities), there is a possibility that multiple ZigBee networks (within range of each other) could use the same 16-bit PAN ID. To resolve potential 16-bit PAN ID conflicts, the ZigBee Alliance created a 64-bit PAN ID.

The 64-bit PAN ID (also called the extended PAN ID), is intended to be a unique, non-duplicated value. When a coordinator starts a network, it can either start a network on a preconfigured 64-bit PAN ID, or it can select a random 64-bit PAN ID. The 64-bit PAN ID is used during joining; if a device has a preconfigured 64-bit PAN ID, it will only join a network with the same 64-bit PAN ID. Otherwise, a device could join any detected PAN and inherit the PAN ID from the network when it joins. The 64-bit PAN ID is included in all ZigBee beacons and is used in 16-bit PAN ID conflict resolution.

Routers and end devices are typically configured to join a network with any 16-bit PAN ID as long as the 64-bit PAN ID is valid. Coordinators typically select a random 16-bit PAN ID for their network.

Since the 16-bit PAN ID only allows up to 65,535 unique values, and since the 16-bit PAN ID is randomly selected, provisions exist in ZigBee to detect if two networks (with different 64-bit PAN IDs) are operating on the same 16-bit PAN ID. If such a conflict is detected, the ZigBee stack can perform PAN ID conflict resolution to change the 16-bit PAN ID of the network in order to resolve the conflict. See the ZigBee specification for details.

To summarize, ZigBee routers and end devices should be configured with the 64-bit PAN ID of the network they want to join. They typically acquire the 16-bit PAN ID when they join a network.

Operating Channel

ZigBee utilizes direct-sequence spread spectrum modulation and operates on a fixed channel. The 802.15.4 PHY defines 16 operating channels (channels 11 to 26) in the 2.4 GHz frequency band. XBee modules support all 16 channels.

ZigBee Application Layers: In Depth

This section provides a more in-depth look at the ZigBee application stack layers (APS, ZDO) including a discussion on ZigBee endpoints, clusters, and profiles. Much of the material in this section can introduce unnecessary details of the ZigBee stack that are not required in many cases.

Skip this section if

- •The XBee does not need to interoperate or talk to non-Digi ZigBee devices
- •The XBee simply needs to send data between devices.

Read this section if

- •The XBee may talk to non-Digi ZigBee devices
- •The XBee requires network management and discovery capabilities of the ZDO layer
- •The XBee needs to operate in a public application profile (smart energy, home automation, etc.)

Application Support Sublayer (APS)

The APS layer in ZigBee adds support for application profiles, cluster IDs, and endpoints.

Application Profiles

Application profiles specify various device descriptions including required functionality for various devices. The collection of device descriptions forms an application profile. Application profiles can be defined as "Public" or "Private" profiles. Private profiles are defined by a manufacturer whereas public profiles are defined, developed,

and maintained by the ZigBee Alliance. Each application profile has a unique profile identifier assigned by the ZigBee Alliance.

Examples of public profiles include:

- Home Automation
- Smart Energy
- Commercial Building Automation

The Smart Energy profile, for example, defines various device types including an energy service portal, load controller, thermostat, in-home display, etc. The Smart Energy profile defines required functionality for each device type. For example, a load controller must respond to a defined command to turn a load on or off. By defining standard communication protocols and device functionality, public profiles allow interoperable ZigBee solutions to be developed by independent manufacturers.

Digi XBee ZB firmware operates on a private profile called the Digi Drop-In Networking profile. However, API mode can be used in many cases to talk to devices in public profiles or non-Digi private profiles. See the API Operations chapter for details.

Clusters

A cluster is an application message type defined within a profile. Clusters are used to specify a unique function, service, or action. For example, the following are some clusters defined in the home automation profile:

- On/Off - Used to switch devices on or off (lights, thermostats, etc.)
- Level Control - Used to control devices that can be set to a level between on and off
- Color Control - Controls the color of color capable devices.

Each cluster has an associated 2-byte cluster identifier (cluster ID). The cluster ID is included in all application transmissions. Clusters often have associated request and response messages. For example, a smart energy gateway (service portal) might send a load control event to a load controller in order to schedule turning on or off an appliance. Upon executing the event, the load controller would send a load control report message back to the gateway.

Devices that operate in an application profile (private or public) must respond correctly to all required clusters. For example, a light switch that will operate in the home automation public profile must correctly implement the On/Off and other required clusters in order to interoperate with other home automation devices. The ZigBee Alliance has defined a ZigBee Cluster Library (ZCL) that contains definitions or various general use clusters that could be implemented in any profile.

XBee modules implement various clusters in the Digi private profile. In addition, the API can be used to send or receive messages on any cluster ID (and profile ID or endpoint). See the Explicit Addressing ZigBee Command API frame in chapter 3 for details.

Endpoints

The APS layer includes supports for endpoints. An endpoint can be thought of as a running application, similar to a TCP/IP port. A single device can support one or more endpoints. Each application endpoint is identified by a 1-byte value, ranging from 1 to 240. Each defined endpoint on a device is tied to an application profile. A device could, for example, implement one endpoint that supports a Smart Energy load controller, and another endpoint that supports other functionality on a private profile.

ZigBee Device Profile

Profile ID 0x0000 is reserved for the ZigBee Device Profile. This profile is implemented on all ZigBee devices. Device Profile defines many device and service discovery features and network management capabilities. Endpoint 0 is a reserved endpoint that supports the ZigBee Device Profile. This endpoint is called the ZigBee Device Objects (ZDO) endpoint.

ZigBee Device Objects (ZDO)

The ZDO (endpoint 0) supports the discovery and management capabilities of the ZigBee Device Profile. A complete listing of all ZDP services is included in the ZigBee specification. Each service has an associated cluster ID.

The XBee ZB firmware allows applications to easily send ZDO messages to devices in the network using the API. See the ZDO Transmissions section in chapter 4 for details.

Coordinator Operation

Forming a Network

The coordinator is responsible for selecting the channel, PAN ID (16-bit and 64-bit), security policy, and stack profile for a network. Since a coordinator is the only device type that can start a network, each ZigBee network must have one coordinator. After the coordinator has started a network, it can allow new devices to join the network. It can also route data packets and communicate with other devices on the network.

To ensure the coordinator starts on a good channel and unused PAN ID, the coordinator performs a series of scans to discover any RF activity on different channels (energy scan) and to discover any nearby operating PANs (PAN scan). The process for selecting the channel and PAN ID are described in the following sections.

Channel Selection

When starting a network, the coordinator must select a "good" channel for the network to operate on. To do this, it performs an energy scan on multiple channels (frequencies) to detect energy levels on each channel. Channels with excessive energy levels are removed from its list of potential channels to start on.

PAN ID Selection

After completing the energy scan, the coordinator scans its list of potential channels (remaining channels after the energy scan) to obtain a list of neighboring PANs. To do this, the coordinator sends a beacon request (broadcast) transmission on each potential channel. All nearby coordinators and routers (that have already joined a ZigBee network) will respond to the beacon request by sending a beacon back to the coordinator. The beacon contains information about the PAN the device is on, including the PAN identifiers (16-bit and 64-bit). This scan (collecting beacons on the potential channels) is typically called an active scan or PAN scan.

After the coordinator completes the channel and PAN scan, it selects a random channel and unused 16-bit PAN ID to start on.

Security Policy

The security policy determines which devices are allowed to join the network, and which device(s) can authenticate joining devices. See chapter 5 for a detailed discussion of various security policies.

Persistent Data

Once a coordinator has started a network, it retains the following information through power cycle or reset events:

- PAN ID
- Operating channel
- Security policy and frame counter values
- Child table (end device children that are joined to the coordinator).
- Binding Table
- Group Table

The coordinator will retain this information indefinitely until it leaves the network. When the coordinator leaves a network and starts a new network, the previous PAN ID, operating channel, and child table data are lost.

XBee ZB Coordinator Startup

The following commands control the coordinator network formation process.

Network formation commands used by the coordinator to form a network.

Command	Description
ID	Used to determine the 64-bit PAN ID. If set to 0 (default), a random 64-bit PAN ID will be selected.
SC	Determines the scan channels bitmask (up to 16 channels) used by the coordinator when forming a network. The coordinator will perform an energy scan on all enabled SC channels. It will then perform a PAN ID scan and then form the network on one of the SC channels.
SD	Set the scan duration period. This value determines how long the coordinator performs an energy scan or PAN ID scan on a given channel.
ZS	Set the ZigBee stack profile for the network.
EE	Enable or disable security in the network.
NK	Set the network security key for the network. If set to 0 (default), a random network security key will be used.
KY	Set the trust center link key for the network. If set to 0 (default), a random link key will be used.
EO	Set the security policy for the network.

Once the coordinator starts a network, the network configuration settings and child table data persist through power cycles as mentioned in the "Persistent Data" section.

When the coordinator has successfully started a network, it

- Allows other devices to join the network for a time (see NJ command)
- Sets AI=0
- Starts blinking the Associate LED
- Sends an API modem status frame ("coordinator started") out the serial port when using API mode.

These behaviors are configurable using the following commands:

Command	Description
NJ	Sets the permit-join time on the coordinator, measured in seconds.
D5	Enables the Associate LED functionality.
LT	Sets the Associate LED blink time when joined. Default is 1 blink per second.

If any of the command values in the network formation commands table changes, the coordinator will leave its current network and start a new network, possibly on a different channel. Note that command changes must be applied (AC or CN command) before taking effect.

Permit Joining

The permit joining attribute on the coordinator is configurable with the NJ command. NJ can be configured to always allow joining, or to allow joining for a short time.

Joining Always Enabled

If NJ=0xFF (default), joining is permanently enabled. This mode should be used carefully. Once a network has been deployed, the application should strongly consider disabling joining to prevent unwanted joins from occurring.

Joining Temporarily Enabled

If NJ < 0xFF, joining will be enabled only for a number of seconds, based on the NJ parameter. The timer is started once the XBee joins a network. Joining will not be re-enabled if the module is power cycled or reset. The following mechanisms can restart the permit-joining timer:

- Changing NJ to a different value (and applying changes with the AC or CN commands)
- Pressing the commissioning button twice
- Issuing the CB command with a parameter of 2

The last two cases enable joining for one minute if NJ is 0x0 or 0xFF. Otherwise, the commissioning button and the CB2 command enable joining for NJ seconds.

Resetting the Coordinator

When the coordinator is reset or power cycled, it checks its PAN ID, operating channel and stack profile against the network configuration settings (ID, CH, ZS). It also verifies the saved security policy against the security configuration settings (EE, NK, KY). If the coordinator's PAN ID, operating channel, stack profile, or security policy is not valid based on its network and security configuration settings, then the coordinator will leave the network and attempt to form a new network based on its network formation command values.

To prevent the coordinator from leaving an existing network, the WR command should be issued after all network formation commands have been configured in order to retain these settings through power cycle or reset events.

Leaving a Network

There are a couple of mechanisms that will cause the coordinator to leave its current PAN and start a new network based on its network formation parameter values. These include the following:

- Change the ID command such that the current 64-bit PAN ID is invalid.
- Change the SC command such that the current channel (CH) is not included in the channel mask.
- Change the ZS or any of the security command values (excluding NK).
- Issue the NR0 command to cause the coordinator to leave.
- Issue the NR1 command to send a broadcast transmission, causing all devices in the network to leave and migrate to a different channel.
- Press the commissioning button 4 times or issue the CB command with a parameter of 4.
- Issue a network leave command.

Note that changes to ID, SC, ZS, and security command values only take effect when changes are applied (AC or CN commands).

Replacing a Coordinator (Security Disabled Only)

In rare occasions, it may become necessary to replace an existing coordinator in a network with a new physical device. If security is not enabled in the network, a replacement XBee coordinator can be configured with the PAN ID (16-bit and 64-bit), channel, and stack profile settings of a running network in order to replace an existing coordinator.

NOTE: Having two coordinators on the same channel, stack profile, and PAN ID (16-bit and 64-bit) can cause problems in the network and should be avoided. When replacing a coordinator, the old coordinator should be turned off before starting the new coordinator.

To replace a coordinator, the following commands should be read from a device on the network:

AT Command	Description
OP	Read the operating 64-bit PAN ID.
OI	Read the operating 16-bit PAN ID.
CH	Read the operating channel.
ZS	Read the stack profile.

Each of the commands listed above can be read from any device on the network. (These parameters will be the same on all devices in the network.) After reading these commands from a device on the network, these parameter values should be programmed into the new coordinator using the following commands.

AT Command	Description
ID	Set the 64-bit PAN ID to match the read OP value.
II	Set the initial 16-bit PAN ID to match the read OI value.
SC	Set the scan channels bitmask to enable the read operating channel (CH command). For example, if the operating channel is 0x0B, set SC to 0x0001. If the operating channel is 0x17, set SC to 0x1000.
ZS	Set the stack profile to match the read ZS value.

Note: II is the initial 16-bit PAN ID. Under certain conditions, the ZigBee stack can change the 16-bit PAN ID of the network. For this reason, the II command cannot be saved using the WR command. Once II is set, the coordinator leaves the network and starts on the 16-bit PAN ID specified by II.

Example: Starting a Coordinator

1. Set CE (Coordinator Enable) to 1, and use the WR command to save the changes.
2. Set SC and ID to the desired scan channels and PAN ID values. (The defaults should suffice.)
3. If SC or ID is changed from the default, issue the WR command to save the changes.
4. If SC or ID is changed from the default, apply changes (make SC and ID changes take effect) either by sending the AC command or by exiting AT command mode.
5. The Associate LED will start blinking once the coordinator has selected a channel and PAN ID.
6. The API Modem Status frame ("Coordinator Started") is sent out the serial port when using API mode.
7. Reading the AI command (association status) will return a value of 0, indicating a successful startup.
8. Reading the MY command (16-bit address) will return a value of 0, the ZigBee-defined 16-bit address of the coordinator.

After startup, the coordinator will allow joining based on its NJ value.

Example: Replacing a Coordinator (Security Disabled)

1. Read the OP, OI, CH, and ZS commands on the running coordinator.
2. Set the CE, ID, SC, and ZS parameters on the new coordinator, followed by WR command to save these parameter values.
3. Turn off the running coordinator.
4. Set the II parameter on the new coordinator to match the read OI value on the old coordinator.
5. Wait for the new coordinator to start (AI=0).

Router Operation

Routers must discover and join a valid ZigBee network before they can participate in a ZigBee network. After a router has joined a network, it can allow new devices to join the network. It can also route data packets and communicate with other devices on the network.

Discovering ZigBee Networks

To discover nearby ZigBee networks, the router performs a PAN (or active) scan, just like the coordinator does when it starts a network. During the PAN scan, the router sends a beacon request (broadcast) transmission on the first channel in its scan channels list. All nearby coordinators and routers operating on that channel (that are already part of a ZigBee network) respond to the beacon request by sending a beacon back to the router. The beacon contains information about the PAN the nearby device is on, including the PAN identifier (PAN ID), and

whether or not joining is allowed. The router evaluates each beacon received on the channel to determine if a valid PAN is found. A router considers a PAN to be valid if the PAN:

- Has a valid 64-bit PAN ID (PAN ID matches ID if ID > 0)
- Has the correct stack profile (ZS command)
- Is allowing joining.

If a valid PAN is not found, the router performs the PAN scan on the next channel in its scan channels list and continues scanning until a valid network is found, or until all channels have been scanned. If all channels have been scanned and a valid PAN was not discovered, all channels will be scanned again.

The ZigBee Alliance requires that certified solutions not send beacon request messages too frequently. To meet certification requirements, the XBee firmware attempts 9 scans per minute for the first 5 minutes, and 3 scans per minute thereafter. If a valid PAN is within range of a joining router, it should typically be discovered within a few seconds.

Joining a Network

Once the router discovers a valid network, it sends an association request to the device that sent a valid beacon requesting a join on the ZigBee network. The device allowing the join then sends an association response frame that either allows or denies the join.

When a router joins a network, it receives a 16-bit address from the device that allowed the join. The 16-bit address is randomly selected by the device that allowed the join.

Authentication

In a network where security is enabled, the router must then go through an authentication process. See the Security chapter for a discussion on security and authentication.

After the router is joined (and authenticated, in a secure network), it can allow new devices to join the network.

Persistent Data

Once a router has joined a network, it retains the following information through power cycle or reset events:

- PAN ID
- Operating channel
- Security policy and frame counter values
- Child table (end device children that are joined to the coordinator).
- Binding Table
- Group Table

The router will retain this information indefinitely until it leaves the network. When the router leaves a network, the previous PAN ID, operating channel, and child table data are lost.

XBee ZB Router Joining

When the router is powered on, if it is not already joined to a valid ZigBee network, it immediately attempts to find and join a valid ZigBee network.

Note: The DJ command can be set to 1 to disable joining. The DJ parameter cannot be written with WR, so a power cycle always clears the DJ setting.

The following commands control the router joining process.

Command	Description
ID	Sets the 64-bit PAN ID to join. Setting ID=0 allows the router to join any 64-bit PAN ID.
SC	Set the scan channels bitmask that determines which channels a router will scan to find a valid network. SC on the router should be set to match SC on the coordinator. For example, setting SC to 0x281 enables scanning on channels 0x0B, 0x12, and 0x14, in that order.
SD	Set the scan duration, or time that the router will listen for beacons on each channel.
ZS	Set the stack profile on the device.
EE	Enable or disable security in the network. This must be set to match the EE value (security policy) of the coordinator.
KY	Set the trust center link key. If set to 0 (default), the link key is expected to be obtained (unencrypted) during joining.

Once the router joins a network, the network configuration settings and child table data persist through power cycles as mentioned in the "Persistent Data" section previously. If joining fails, the status of the last join attempt can be read in the AI command register.

If any of the above command values change, when command register changes are applied (AC or CN commands), the router will leave its current network and attempt to discover and join a new valid network.

When a ZB router has successfully joined a network, it:

- Allows other devices to join the network for a time
- Sets AI=0
- Starts blinking the Associate LED
- Sends an API modem status frame ("associated") out the serial port when using API mode.

These behaviors are configurable using the following commands:

Command	Description
NJ	Sets the permit-join time on the router, or the time that it will allow new devices to join the network, measured in seconds. If NJ=0xFF, permit joining will always be enabled.
D5	Enables the Associate LED functionality.
LT	Sets the Associate LED blink time when joined. Default is 2 blinks per second (router).

Permit Joining

The permit joining attribute on the router is configurable with the NJ command. NJ can be configured to always allow joining, or to allow joining for a short time.

Joining Always Enabled

If NJ=0xFF (default), joining is permanently enabled. This mode should be used carefully. Once a network has been deployed, the application should strongly consider disabling joining to prevent unwanted joins from occurring.

Joining Temporarily Enabled

If NJ < 0xFF, joining will be enabled only for a number of seconds, based on the NJ parameter. The timer is started once the XBee joins a network. Joining will not be re-enabled if the module is power cycled or reset. The following mechanisms can restart the permit-joining timer:

- Changing NJ to a different value (and applying changes with the AC or CN commands)
- Pressing the commissioning button twice
- Issuing the CB command with a parameter of 2 (software emulation of a 2 button press)

- Causing the router to leave and rejoin the network.

The middle two cases enable joining for one minute if NJ is 0x0 or 0xFF. Otherwise, the commissioning button and the CB2 command enable joining for NJ seconds.

Router Network Connectivity

Once a router joins a ZigBee network, it remains connected to the network on the same channel and PAN ID as long as it is not forced to leave. (See “Leaving a Network” section for details.) If the scan channels (SC), PAN ID (ID) and security settings (EE, KY) do not change after a power cycle, the router will remain connected to the network after a power cycle.

If a router may physically move out of range of the network it initially joined, the application should include provisions to detect if the router can still communicate with the original network. If communication with the original network is lost, the application may choose to force the router to leave the network (see Leaving a Network section below for details). The XBee firmware includes two provisions to automatically detect the presence of a network, and leave if the check fails.

Power-On Join Verification

The JV command (join verification) enables the power-on join verification check. If enabled, the XBee will attempt to discover the 64-bit address of the coordinator when it first joins a network. Once it has joined, it will also attempt to discover the 64-bit address of the coordinator after a power cycle event. If 3 discovery attempts fail, the router will leave the network and try to join a new network. Power-on join verification is disabled by default (JV defaults to 0).

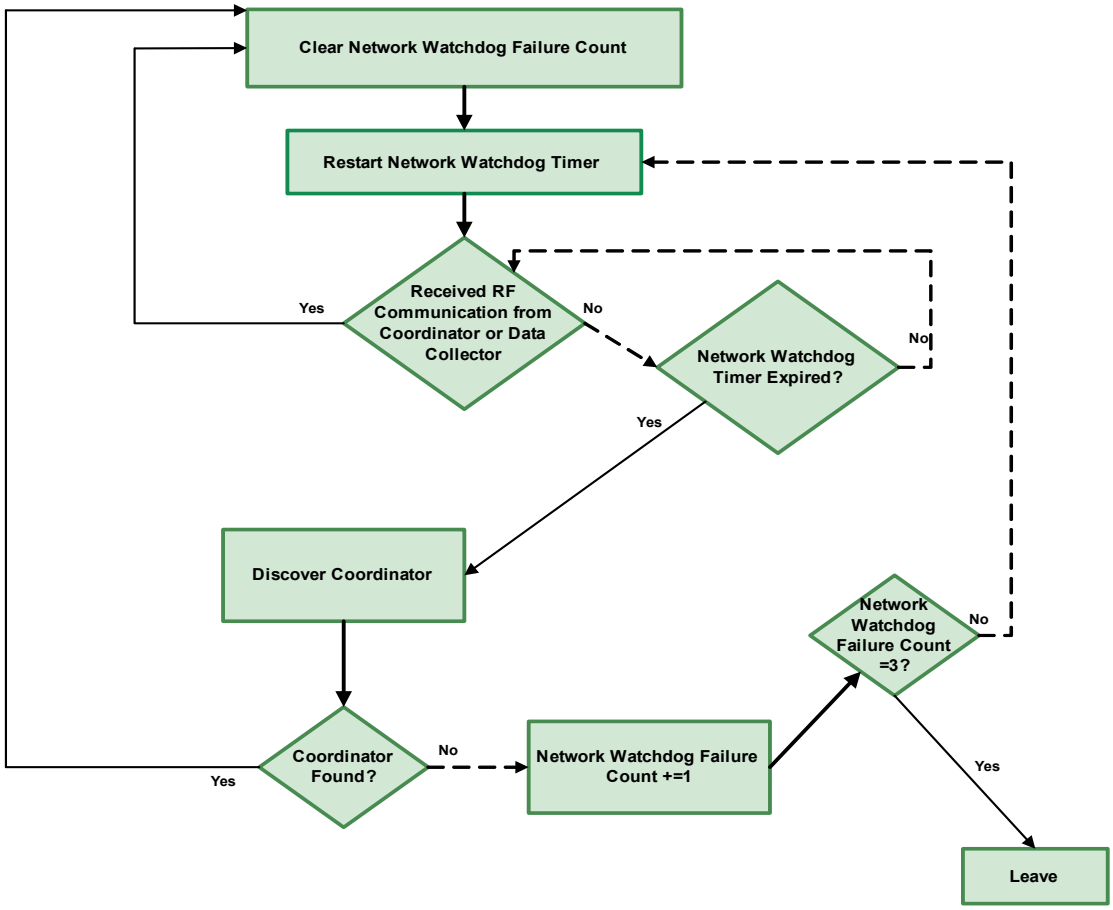
Network Watchdog

The NW command (network watchdog timeout) can be used for a powered router to periodically check for the presence of a coordinator to verify network connectivity. The NW command specifies a timeout in minutes where the router must receive communication from the coordinator or data collector. The following events restart the network watchdog timer:

- RF data received from the coordinator
- RF data sent to the coordinator and an acknowledgment was received
- Many-to-one route request was received (from any device)
- Changing the value of NW.

If the watchdog timer expires (no valid data received for NW time), the router will attempt to discover the 64-bit address of the coordinator. If the address cannot be discovered, the router records one watchdog timeout. Once three consecutive network watchdog timeouts have expired (3 * NW) and the coordinator has not responded to the address discovery attempts, the router will leave the network and attempt to join a new network. Anytime a router receives valid data from the coordinator or data collector, it will clear the watchdog timeouts counter and restart the watchdog timer. The watchdog timer (NW command) is settable to several days. The network watchdog feature is disabled by default (NW defaults to 0).

Network Watchdog Behavior



Leaving a Network

There are a couple of mechanisms that will cause the router to leave its current PAN and attempt to discover and join a new network based on its network joining parameter values.

These include the following:

- Change the ID command such that the current 64-bit PAN ID is invalid.
- Change the SC command such that the current channel (CH) is not included in the channel mask.
- Change the ZS or any of the security command values.
- Issue the NR0 command to cause the router to leave.
- Issue the NR1 command to send a broadcast transmission, causing all devices in the network to leave and migrate to a different channel.
- Press the commissioning button 4 times or issue the CB command with a parameter of 4.
- Issue a network leave command.

Note that changes to ID, SC, ZS, and security command values only take effect when changes are applied (AC or CN commands).

Resetting the Router

When the router is reset or power cycled, it checks its PAN ID, operating channel and stack profile against the network configuration settings (ID, SC, ZS). It also verifies the saved security policy is valid based on the security configuration commands (EE, KY). If the router's PAN ID, operating channel, stack profile, or security policy is invalid, the router will leave the network and attempt to join a new network based on its network joining command values.

To prevent the router from leaving an existing network, the WR command should be issued after all network joining commands have been configured in order to retain these settings through power cycle or reset events.

Example: Joining a Network

After starting a coordinator (that is allowing joins), the following steps will cause a router to join the network:

1. Set ID to the desired 64-bit PAN ID, or to 0 to join any PAN.
2. Set SC to the list of channels to scan to find a valid network.
3. If SC or ID is changed from the default, apply changes (make SC and ID changes take effect) by issuing the AC or CN command.
4. The Associate LED will start blinking once the router has joined a PAN.
5. If the Associate LED is not blinking, the AI command can be read to determine the cause of join failure.
6. Once the router has joined, the OP and CH commands will indicate the operating 64-bit PAN ID and channel the router joined.
7. The MY command will reflect the 16-bit address the router received when it joined.
8. The API Modem Status frame ("Associated") is sent out the serial port when using API mode.
9. The joined router will allow other devices to join for a time based on its NJ setting.

End Device Operation

Similar to routers, end devices must also discover and join a valid ZigBee network before they can participate in a network. After an end device has joined a network, it can communicate with other devices on the network. Since end devices are intended to be battery powered and therefore support low power (sleep) modes, end devices cannot allow other devices to join, nor can they route data packets.

Discovering ZigBee Networks

End devices go through the same process as routers to discover networks by issuing a PAN scan. After sending the broadcast beacon request transmission, the end device listens for a short time in order to receive beacons sent by nearby routers and coordinators on the same channel. The end device evaluates each beacon received on the channel to determine if a valid PAN is found. An end device considers a PAN to be valid if the PAN:

- Has a valid 64-bit PAN ID (PAN ID matches ID if ID > 0)
- Has the correct stack profile (ZS command)
- Is allowing joining
- Has capacity for additional end devices (see End Device Capacity section below).

If a valid PAN is not found, the end device performs the PAN scan on the next channel in its scan channels list and continues this process until a valid network is found, or until all channels have been scanned. If all channels have been scanned and a valid PAN was not discovered, the end device may enter a low power sleep state and scan again later.

If scanning all SC channels fails to discover a valid PAN, XBee ZB modules will attempt to enter a low power state and will retry scanning all SC channels after the module wakes from sleeping. If the module cannot enter a low power state, it will retry scanning all channels, similar to the router. To meet ZigBee Alliance requirements, the end device will attempt up to 9 scans per minute for the first 5 minutes, and 3 scans per minute thereafter.

Note: The XBee ZB end device will not enter sleep until it has completed scanning all SC channels for a valid network.

Joining a Network

Once the end device discovers a valid network, it joins the network, similar to a router, by sending an association request (to the device that sent a valid beacon) to request a join on the ZigBee network. The device allowing the join then sends an association response frame that either allows or denies the join.

When an end device joins a network, it receives a 16-bit address from the device that allowed the join. The 16-bit address is randomly selected by the device that allowed the join.

Parent Child Relationship

Since an end device may enter low power sleep modes and not be immediately responsive, the end device relies on the device that allowed the join to receive and buffer incoming messages in its behalf until it is able to wake and receive those messages. The device that allowed an end device to join becomes the parent of the end device, and the end device becomes a child of the device that allowed the join.

End Device Capacity

Routers and coordinators maintain a table of all child devices that have joined called the child table. This table is a finite size and determines how many end devices can join. If a router or coordinator has at least one unused entry in its child table, the device is said to have end device capacity. In other words, it can allow one or more additional end devices to join. ZigBee networks should have sufficient routers to ensure adequate end device capacity.

The initial release of software on this platform supports up to 20 end devices when configured as a coordinator or a router.

In ZB firmware, the NC command (number of remaining end device children) can be used to determine how many additional end devices can join a router or coordinator. If NC returns 0, then the router or coordinator device has no more end device capacity. (Its child table is full.)

Also of note, since routers cannot sleep, there is no equivalent need for routers or coordinators to track joined routers. Therefore, there is no limit to the number of routers that can join a given router or coordinator device. (There is no "router capacity" metric.)

Authentication

In a network where security is enabled, the end device must then go through an authentication process. See chapter 5 for a discussion on security and authentication.

Persistent Data

The end device can retain its PAN ID, operating channel, and security policy information through a power cycle. However, since end devices rely heavily on a parent, the end device does an orphan scan to try and contact its parent. If the end device does not receive an orphan scan response (called a coordinator realignment command), it will leave the network and try to discover and join a new network. When the end device leaves a network, the previous PAN ID and operating channel settings are lost.

Orphan Scans

When an end device comes up from a power cycle, it performs an orphan scan to verify it still has a valid parent. The orphan scan is sent as a broadcast transmission and contains the 64-bit address of the end device. Nearby routers and coordinator devices that receive the broadcast check their child tables for an entry that contains the end device's 64-bit address. If an entry is found with a matching 64-bit address, the device sends a coordinator realignment command to the end device that includes the end device's 16-bit address, 16-bit PAN ID, operating channel, and the parent's 64-bit and 16-bit addresses.

If the orphaned end device receives a coordinator realignment command, it is considered joined to the network. Otherwise, it will attempt to discover and join a valid network.

XBee ZB End Device Joining

When an end device is powered on, if it is not joined to a valid ZigBee network, or if the orphan scan fails to find a parent, it immediately attempts to find and join a valid ZigBee network.

Note: The DJ command can be set to 1 to disable joining. The DJ parameter cannot be written with WR, so a power cycle always clears the DJ setting.

Similar to a router, the following commands control the end device joining process.

Network joining commands used by an end device to join a network.

Command	Description
ID	Sets the 64-bit PAN ID to join. Setting ID=0 allows the router to join any 64-bit PAN ID.
SC	Set the scan channels bitmask that determines which channels an end device will scan to find a valid network. SC on the end device should be set to match SC on the coordinator and routers in the desired network. For example, setting SC to 0x281 enables scanning on channels 0x0B, 0x12, and 0x14, in that order.
SD	Set the scan duration, or time that the end device will listen for beacons on each channel.
ZS	Set the stack profile on the device.
EE	Enable or disable security in the network. This must be set to match the EE value (security policy) of the coordinator.
KY	Set the trust center link key. If set to 0 (default), the link key is expected to be obtained (unencrypted) during joining.

Once the end device joins a network, the network configuration settings can persist through power cycles as mentioned in the "Persistent Data" section previously. If joining fails, the status of the last join attempt can be read in the AI command register.

If any of these command values changes, when command register changes are applied, the end device will leave its current network and attempt to discover and join a new valid network.

When a ZB end device has successfully started a network, it

- Sets AI=0
- Starts blinking the Associate LED
- Sends an API modem status frame ("associated") out the serial port when using API mode
- Attempts to enter low power modes.

These behaviors are configurable using the following commands:

Command	Description
D5	Enables the Associate LED functionality.
LT	Sets the Associate LED blink time when joined. Default is 2 blinks per second (end devices).
SM, SP, ST, SN, SO	Parameters that configure the sleep mode characteristics. (See Managing End Devices chapter for details.)

Parent Connectivity

The XBee ZB end device sends regular poll transmissions to its parent when it is awake. These poll transmissions query the parent for any new received data packets. The parent always sends a MAC layer acknowledgment back to the end device. The acknowledgment indicates whether the parent has data for the end device or not.

If the end device does not receive an acknowledgment for 3 consecutive poll requests, it considers itself disconnected from its parent and will attempt to discover and join a valid ZigBee network. See "Managing End Devices" chapter for details.

Resetting the End Device

When the end device is reset or power cycled, if the orphan scan successfully locates a parent, the end device then checks its PAN ID, operating channel and stack profile against the network configuration settings (ID, SC, ZS). It also verifies the saved security policy is valid based on the security configuration commands (EE, KY). If the end device's PAN ID, operating channel, stack profile, or security policy is invalid, the end device will leave the network and attempt to join a new network based on its network joining command values.

To prevent the end device from leaving an existing network, the WR command should be issued after all network joining commands have been configured in order to retain these settings through power cycle or reset events.

Leaving a Network

There are a couple of mechanisms that will cause the router to leave its current PAN and attempt to discover and join a new network based on its network joining parameter values. These include the following:

- The ID command changes such that the current 64-bit PAN ID is invalid.
- The SC command changes such that the current operating channel (CH) is not included in the channel mask.
- The ZS or any of the security command values change.
- The NR0 command is issued to cause the end device to leave.
- The NR1 command is issued to send a broadcast transmission, causing all devices in the network to leave and migrate to a different channel.
- The commissioning button is pressed 4 times or the CB command is issued with a parameter of 4.
- The end device's parent is powered down or the end device is moved out of range of the parent such that the end device fails to receive poll acknowledgment messages.

Note that changes to command values only take effect when changes are applied (AC or CN commands).

Example: Joining a Network

After starting a coordinator (that is allowing joins), the following steps will cause an XBee end device to join the network:

1. Set ID to the desired 64-bit PAN ID, or to 0 to join any PAN.
2. Set SC to the list of channels to scan to find a valid network.
3. If SC or ID is changed from the default, apply changes (make SC and ID changes take effect) by issuing the AC or CN command.
4. The Associate LED will start blinking once the end device has joined a PAN.
5. If the Associate LED is not blinking, the AI command can be read to determine the cause of join failure.
6. Once the end device has joined, the OP and CH commands will indicate the operating 64-bit PAN ID and channel the end device joined.
7. The MY command will reflect the 16-bit address the router received when it joined.
8. The API Modem Status frame ("Associated") is sent out the serial port when using API mode.
9. The joined end device will attempt to enter low power sleep modes based on its sleep configuration commands (SM, SP, SN, ST, SO).

Channel Scanning

As mentioned previously, routers and end devices must scan one or more channels to discover a valid network to join. When a join attempt begins, the XBee sends a beacon request transmission on the lowest channel specified in the SC (scan channels) command bitmask. If a valid PAN is found on the channel, the XBee will attempt to join the PAN on that channel. Otherwise, if a valid PAN is not found on the channel, it will attempt scanning on the next higher channel in the SC command bitmask. The XBee will continue to scan each channel (from lowest to highest) in the SC bitmask until a valid PAN is found or all channels have been scanned. Once all channels have been scanned, the next join attempt will start scanning on the lowest channel specified in the SC command bitmask.

For example, if the SC command is set to 0x400F, the XBee would start scanning on channel 11 (0x0B) and scan until a valid beacon is found, or until channels 11, 12, 13, 14, and 25 have been scanned (in that order).

Once an XBee router or end device joins a network on a given channel, if the XBee is told to leave (see "Leaving a Network" section), it will leave the channel it joined on and continue scanning on the next higher channel in the SC bitmask.

For example, if the SC command is set to 0x400F, and the XBee joins a PAN on channel 12 (0x0C), if the XBee leaves the channel, it will start scanning on channel 13, followed by channels 14 and 25 if a valid network is not found. Once all channels have been scanned, the next join attempt will start scanning on the lowest channel specified in the SC command bitmask.

Managing Multiple ZigBee Networks

In some applications, multiple ZigBee networks may exist in proximity of each other. The application may need provisions to ensure the XBee joins the desired network. There are a number of features in ZigBee to manage joining among multiple networks. These include the following:

- PAN ID Filtering
- Preconfigured Security Keys
- Permit Joining
- Application Messaging

PAN ID Filtering

The XBee can be configured with a fixed PAN ID by setting the ID command to a non-zero value. If the PAN ID is set to a non-zero value, the XBee will only join a network with the same PAN ID.

Pre-configured Security Keys

Similar to PAN ID filtering, this method requires a known security key be installed on a router to ensure it will join a ZigBee network with the same security key. If the security key (KY command) is set to a non-zero value, and if security is enabled (EE command), an XBee router or end device will only join a network with the same security key.

Permit Joining

The Permit Joining parameter can be disabled in a network to prevent unwanted devices from joining. When a new device must be added to a network, permit-joining can be enabled for a short time on the desired network. In the XBee firmware, joining is disabled by setting the NJ command to a value less than 0xFF on all routers and coordinator devices. Joining can be enabled for a short time using the commissioning push-button (see Network Commissioning chapter for details) or the CB command.

Application Messaging

If the above mechanisms are not feasible, the application could build in a messaging framework between the coordinator and devices that join its network. For example, the application code in joining devices could send a transmission to the coordinator after joining a network, and wait to receive a defined reply message. If the application does not receive the expected response message after joining, the application could force the XBee to leave and continue scanning (see NR parameter).

4. Transmission, Addressing, and Routing

Addressing

All ZigBee devices have two different addresses, a 64-bit and a 16-bit address. The characteristics of each are described below.

64-bit Device Addresses

The 64-bit address is a unique device address assigned during manufacturing. This address is unique to each physical device. The 64-bit address includes a 3-byte Organizationally Unique Identifier (OUI) assigned by the IEEE. The 64-bit address is also called the extended address.

16-bit Device Addresses

A device receives a 16-bit address when it joins a ZigBee network. For this reason, the 16-bit address is also called the "network address". The 16-bit address of 0x0000 is reserved for the coordinator. All other devices receive a randomly generated address from the router or coordinator device that allows the join. The 16-bit address can change under certain conditions:

- An address conflict is detected where two devices are found to have the same 16-bit address
- A device leaves the network and later joins (it can receive a different address)

All ZigBee transmissions are sent using the source and destination 16-bit addresses. The routing tables on ZigBee devices also use 16-bit addresses to determine how to route data packets through the network. However, since the 16-bit address is not static, it is not a reliable way to identify a device.

To solve this problem, the 64-bit destination address is often included in data transmissions to guarantee data is delivered to the correct destination. The ZigBee stack can discover the 16-bit address, if unknown, before transmitting data to a remote.

Application Layer Addressing

ZigBee devices can support multiple application profiles, cluster IDs, and endpoints. (See "ZigBee Application Layers - In Depth" in chapter 3.) Application layer addressing allows data transmissions to be addressed to specific profile IDs, cluster IDs, and endpoints. Application layer addressing is useful if an application must

- Interoperate with other ZigBee devices outside of the Digi application profile
- Utilize service and network management capabilities of the ZDO
- Operate on a public application profile such as Home Controls or Smart Energy.

API mode provides a simple yet powerful interface that can easily send data to any profile ID, endpoint, and cluster ID combination on any device in a ZigBee network.

Data Transmission

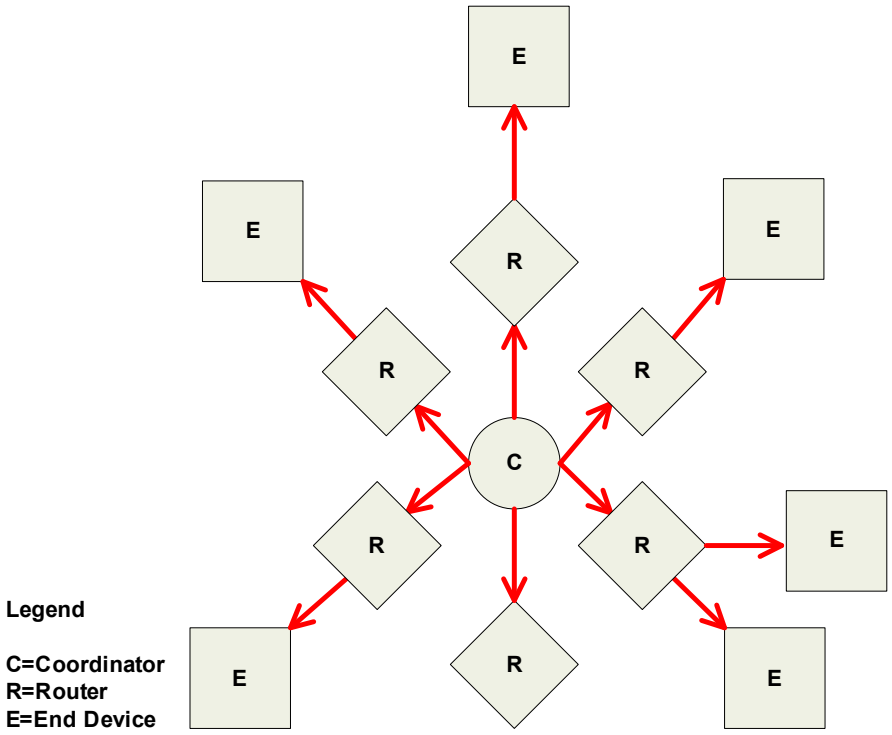
ZigBee data packets can be sent as either unicast or broadcast transmissions. Unicast transmissions route data from one source device to one destination device, whereas broadcast transmissions are sent to many or all devices in the network.

Broadcast Transmissions

Broadcast transmissions within the ZigBee protocol are intended to be propagated throughout the entire network such that all nodes receive the transmission. To accomplish this, the coordinator and all routers that receive a broadcast transmission will retransmit the packet three times.

Note: when a router or coordinator delivers a broadcast transmission to an end device child, the transmission is only sent once (immediately after the end device wakes and polls the parent for any new data). See Parent Operation section in chapter 6 for details.

Broadcast Data Transmission



Each node that transmits the broadcast will also create an entry in a local broadcast transmission table. This entry is used to keep track of each received broadcast packet to ensure the packets are not endlessly transmitted. Each entry persists for 8 seconds. The broadcast transmission table holds 8 entries.

For each broadcast transmission, the ZigBee stack must reserve buffer space for a copy of the data packet. This copy is used to retransmit the packet as needed. Large broadcast packets will require more buffer space. This information on buffer space is provided for general knowledge; the user does not and cannot change any buffer spacing. Buffer spacing is handled automatically by the XBee module.

Since broadcast transmissions are retransmitted by each device in the network, broadcast messages should be used sparingly.

Unicast Transmissions

Unicast transmissions are sent from one source device to another destination device. The destination device could be an immediate neighbor of the source, or it could be several hops away. Unicast transmissions that are sent along a multiple hop path require some means of establishing a route to the destination device. See the "RF Packet Routing" section in chapter 4 for details.

Address Resolution

As mentioned previously, each device in a ZigBee network has both a 16-bit (network) address and a 64-bit (extended) address. The 64-bit address is unique and assigned to the device during manufacturing, and the 16-bit address is obtained after joining a network. The 16-bit address can also change under certain conditions.

When sending a unicast transmission, the ZigBee network layer uses the 16-bit address of the destination and each hop to route the data packet. If the 16-bit address of the destination is not known, the ZigBee stack includes a discovery provision to automatically discover the destination device's 16-bit address before routing the data.

To discover a 16-bit address of a remote, the device initiating the discovery sends a broadcast address discovery transmission. The address discovery broadcast includes the 64-bit address of the remote device whose 16-bit address is being requested. All nodes that receive this transmission check the 64-bit address in the payload and compare it to their own 64-bit address. If the addresses match, the device sends a response packet back to the initiator. This response includes the remote's 16-bit address. When the discovery response is received, the initiator will then transmit the data.

Address Table

Each ZigBee device maintains an address table that maps a 64-bit address to a 16-bit address. When a transmission is addressed to a 64-bit address, the ZigBee stack searches the address table for an entry with a matching 64-bit address, in hopes of determining the destination's 16-bit address. If a known 16-bit address is not found, the ZigBee stack will perform address discovery to discover the device's current 16-bit address.

Sample Address Table

64-bit Address	16-bit Address
0013 A200 4000 0001	0x4414
0013 A200 400A 3568	0x1234
0013 A200 4004 1122	0xC200
0013 A200 4002 1123	0xFFFE (unknown)

The XBee modules can store up to 10 address table entries. For applications where a single device (e.g. coordinator) may send unicast transmissions to more than 10 devices, the application should implement an address table to store the 16-bit and 64-bit addresses for each remote device. Any XBee that will send data to more than 10 remotes should also use API mode. The application can then send both the 16-bit and 64-bit addresses to the XBee in the API transmit frames which will significantly reduce the number of 16-bit address discoveries and greatly improve data throughput.

If an application will support an address table, the size should ideally be larger than the maximum number of destination addresses the device will communicate with. Each entry in the address table should contain a 64-bit destination address and its last known 16-bit address.

When sending a transmission to a destination 64-bit address, the application should search the address table for a matching 64-bit address. If a match is found, the 16-bit address should be populated into the 16-bit address field of the API frame. If a match is not found, the 16-bit address should be set to 0xFFFE (unknown) in the API transmit frame.

The API provides indication of a remote device's 16-bit address in the following frames:

- All receive data frames

  Rx Data (0x90)

  Rx Explicit Data (0x91)

  I/O Sample Data (0x92)

  Node Identification Indicator (0x95)

  Route Record Indicator (0xA1)

  etc.

- Transmit status frame (0x8B)

Group Table

Each router and the coordinator maintain a persistent group table. Each entry contains an endpoint value, a two byte group ID, and an optional name string of zero to 16 ASCII characters, and an index into the binding table. More than one endpoint may be associated with a group ID, and more than one group ID may be associated with a given endpoint. The capacity of the group table is 16 entries.

The application should always update the 16-bit address in the address table when one of these frames is received to ensure the table has the most recently known 16-bit address. If a transmission failure occurs, the application should set the 16-bit address in the table to 0xFFFE (unknown).

Binding Transmissions

Binding transmissions use indirect addressing to send one or more messages to other destination devices. An Explicit Addressing ZigBee Command Frame (0x11) using the Indirect Tx Option (0x04) is treated as a binding transmission request.

Address Resolution

The source endpoint and cluster ID values of a binding transmission are used as keys to lookup matching binding table entries. For each matching binding table entry, the type field of the entry indicates whether a unicast or a multicast message should be sent.

In the case of a unicast entry, the transmission request is updated with the Destination Endpoint and MAC Address, and unicast to its destination. In the case of a multicast entry, the message is updated using the two least significant bytes of the Destination MAC Address as the groupID, and multicast to its destination(s).

Binding Table

Each router and the coordinator maintain a persistent binding table to map source endpoint and cluster ID values into 64 bit destination address and endpoint values. The capacity of the binding table is 16 entries.

Multicast Transmissions

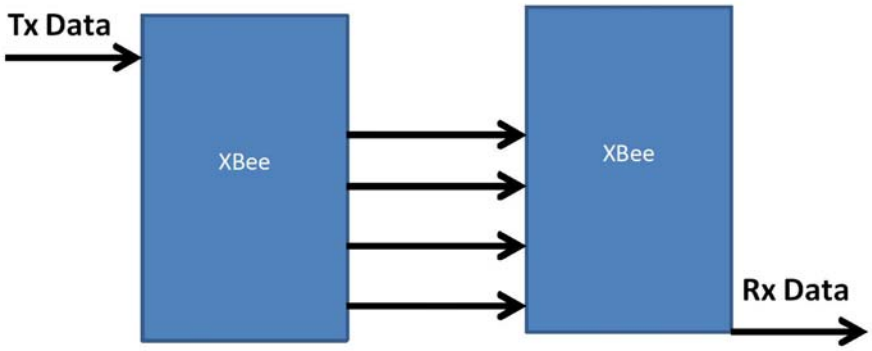
Multicast transmissions are used to broadcast a message to destination devices which have active endpoints associated with a common group ID. An explicit transmit request frame (0x11) using the Multicast Tx Option (0x08) is treated as a multicast transmission request.

Address Resolution

The 64 bit destination address value does not matter and it is recommended it be set to 0xFFFFFFFFFFFFFFFF. The 16 bit destination address value should be set to the destination groupID.

Fragmentation

Each unicast transmission may support up to 84 bytes of RF payload. (Enabling security or using source routing can reduce this number. See the NP command for details.) However, the XBee ZB firmware supports a ZigBee feature called fragmentation that allows a single large data packet to be broken up into multiple RF transmissions and reassembled by the receiver before sending data out its serial port. This is shown in the image below.



The transmit frame can include up to 255 bytes of data, which will be broken up into multiple transmissions and reassembled on the receiving side. If one or more of the fragmented messages are not received by the receiving device, the receiver will drop the entire message, and the sender will indicate a transmission failure in the Tx Status API frame.

Applications that do not wish to use fragmentation should avoid sending more than the maximum number of bytes in a single RF transmission. See the "Maximum RF Payload Size" section for details.

If RTS flow control is enabled on the receiving module (using the D6 command) and a fragmented message is received, then RTS flow control will be ignored.

Data Transmission Examples

AT Firmware

To send a data packet in transparent mode, the DH and DL commands must be set to match the 64-bit address of the destination device. DH must match the upper 4-bytes, and DL must match the lower 4 bytes. Since the coordinator always receives a 16-bit address of 0x0000, a 64-bit address of 0x0000000000000000 is defined as the coordinator's address (in ZB firmware). The default values of DH and DL are 0x00, which sends data to the coordinator.

Example 1: Send a transmission to the coordinator.

(In this example, a '\r' refers to a carriage return character.)

A router or end device can send data in two ways. First, set the destination address (DH and DL commands) to 0x00.

1. Enter command mode ('+++')
2. After receiving an OK\r, issue the following commands:
   a. ATDH0\r
   b. ATDL0\r
   c. ATCN\r
3. Verify that each of the 3 commands returned an OK\r response.
4. After setting these command values, all serial characters will be sent as a unicast transmission to the coordinator.

Alternatively, if the coordinator's 64-bit address is known, DH and DL can be set to the coordinator's 64-bit address. Suppose the coordinator's address is 0x0013A200404A2244.

1. Enter command mode ('+++')
2. After receiving an OK\r, issue the following commands:
   a. ATDH13A200\r
   b. ATDL404A2244\
   c. ATCN\r
3. Verify that each of the 3 commands returned an OK\r response.

4. After setting these command values, all serial characters will be sent as a unicast transmission to the coordinator.

API Firmware

Use the transmit request, or explicit transmit request frame (0x10 and 0x11 respectively) to send data to the coordinator. The 64-bit address can either be set to 0x0000000000000000, or to the 64-bit address of the coordinator. The 16-bit address should be set to 0xFFFE when using the 64-bit address of all 0x00s.

To send an ascii "1" to the coordinator's 0x00 address, the following API frame can be used:

7E 00 0F 10 01 0000 0000 0000 0000 FFFE 00 00 31 C0

If the explicit transmit frame is used, the cluster ID should be set to 0x0011, the profile ID to 0xC105, and the source and destination endpoints to 0xE8 (recommended defaults for data transmissions in the Digi profile.) The same transmission could be sent using the following explicit transmit frame:

7E 00 15 11 01 0000 0000 0000 0000 FFFE E8 E8 0011 C105 00 00 31 18

Notice the 16-bit address is set to 0xFFFE. This is required when sending to a 64-bit address of 0x00s.

Now suppose the coordinator's 64-bit address is 0x0013A200404A2244. The following transmit request API frame (0x10) will send an ASCII "1" to the coordinator:

7E 00 0F 10 01 0013 A200 404A 2244 0000 0000 31 18

Example 2: Send a broadcast transmission.

(In this example, a '\r' refers to a carriage return character.)

Perform the following steps to configure a broadcast transmission:

1. Enter command mode ('+++')
2. After receiving an OK\r, issue the following commands:
   a. ATDH0\r
   b. ATDLffff\r
   c. ATCN\r
3. Verify that each of the 3 commands returned an OK\r response
4. After setting these command values, all serial characters will be sent as a broadcast transmission.

API Firmware

This example will use the transmit request API frame (0x10) to send an ASCII "1" in a broadcast transmission.

To send an ascii "1" as a broadcast transmission, the following API frame can be used:

7E 00 0F 10 01 0000 0000 0000 FFFF FFFE 00 00 31 C2

Notice the destination 16-bit address is set to 0xFFFE for broadcast transmissions.

Example 3: Send an indirect (binding) transmission.

This example will use the explicit transmit request frame (0x11) to send a transmission using indirect addressing through the binding table. It assumes the binding table has already been set up to map a source endpoint of 0xE7 and cluster ID of 0x0011 to a destination endpoint and 64 bit destination address. The message data is a manufacturing specific profile message using profile ID 0xC105, command ID 0x00, a ZCL Header of 151E10, transaction number EE, and a ZCL payload of 000102030405.

7E 001E 11 e4 FFFFFFFFFFFFFFFF FFFE E7 FF 0011 C105 00 04 151E10EE000102030405 14

Note: The 64 bit destination address has been set to all 0xFF values, and the destination endpoint set to 0xFF. The Tx Option 0x04 indicates indirect addressing is to be used. The 64 bit destination address and destination endpoint will be filled in by looking up data associated with binding table entries which match Example 5: Send a multicast (group ID) broadcast.

Example 4: Send a multicast (group ID) broadcast.

This example will use the explicit transmit request frame (0x11) to send a transmission using multicasting. It assumes the destination devices already have their group tables set up to associate an active endpoint with the group ID (0x1234) of the multicast transmission. The message data is a manufacturing specific profile message using profile ID 0xC105, command ID 0x00, a ZCL Header of 151E10, transaction number EE, and a ZCL payload of 000102030405.

7E 001E 11 01 FFFFFFFFFFFFFFFF 1234 E6 FE 0001 C105 00 08 151E10EE000102030405 BC

Note: The 64 bit destination address has been set to all 0xFF values, and the destination endpoint set to 0xFE. The Tx Option 0x08 indicates multicast (group) addressing is to be used.

RF Packet Routing

Unicast transmissions may require some type of routing. ZigBee includes several different ways to route data, each with its own advantages and disadvantages. These are summarized in the table below.

Routing Approach	Description	When to Use
Ad hoc On-demand Distance Vector (AODV) Mesh Routing	Routing paths are created between source and destination, possibly traversing multiple nodes ("hops"). Each device knows who to send data to next to eventually reach the destination	Use in networks that will not scale beyond about 40 destination devices.
Many-to-One Routing	A single broadcast transmission configures reverse routes on all devices into the device that sends the broadcast	Useful when many remote devices must send data to a single gateway or collector device.
Source Routing	Data packets include the entire route the packet should traverse to get from source to destination	Improves routing efficiency in large networks (over 40 remote devices)

Note – End devices do not make use of these routing protocols. Rather, an end device sends a unicast transmission to its parent and allows the parent to route the data packet in its behalf.

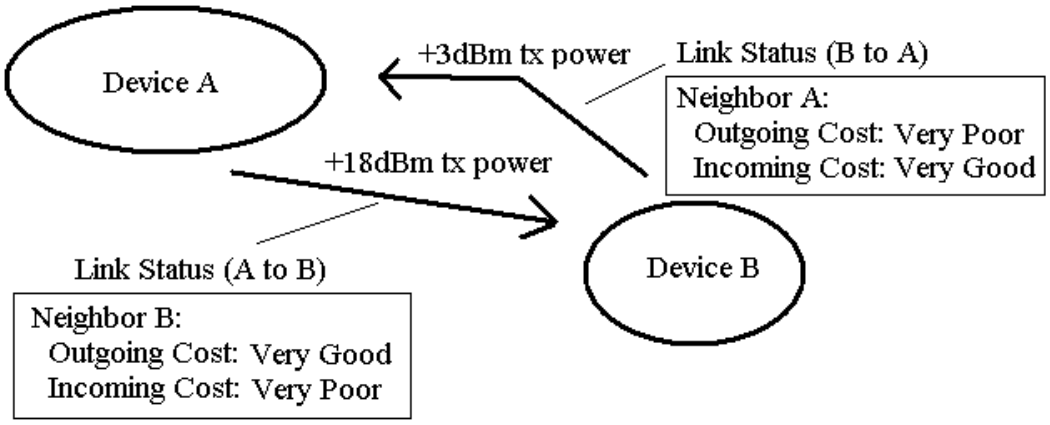
Note – A network cannot revert from Many-to-One routing to AODV routing without first doing a network reset (NR).

Link Status Transmission

Before discussing the various routing protocols, it is worth understanding the primary mechanism in ZigBee for establishing reliable bi-directional links. This mechanism is especially useful in networks that may have a mixture of devices with varying output power and/or receiver sensitivity levels.

Each coordinator or router device periodically sends a link status message. This message is sent as a 1-hop broadcast transmission, received only by one-hop neighbors. The link status message contains a list of neighboring devices and incoming and outgoing link qualities for each neighbor. Using these messages, neighboring devices can determine the quality of a bi-directional link with each neighbor and use that information to select a route that works well in both directions.

For example, consider a network of two neighboring devices that send periodic link status messages. Suppose that the output power of device A is +18dBm, and the output power of device B is +3dBm (considerably less than the output power of device A). The link status messages might indicate the following:



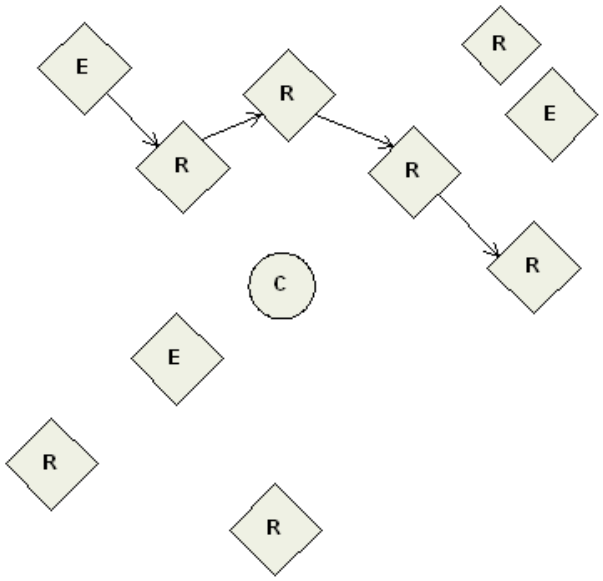
This mechanism enables devices A and B to recognize that the link is not reliable in both directions and select a different neighbor when establishing routes. (Such links are called asymmetric links, meaning the link quality is not similar in both directions.)

When a router or coordinator device powers on, it sends link status messages every couple seconds to attempt to discover link qualities with its neighbors quickly. After being powered on for some time, the link status messages are sent at a much slower rate (about every 3-4 times per minute).

AODV Mesh Routing

ZigBee employs mesh routing to establish a route between the source device and the destination. Mesh routing allows data packets to traverse multiple nodes (hops) in a network to route data from a source to a destination. Routers and coordinators can participate in establishing routes between source and destination devices using a process called route discovery. The Route discovery process is based on the AODV (Ad-hoc On-demand Distance Vector routing) protocol.

Sample Transmission Through a Mesh Network



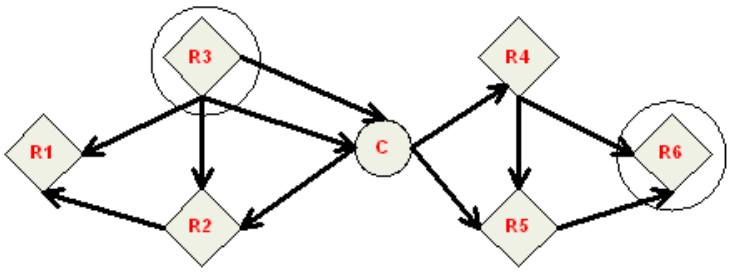
AODV (Ad-hoc On-demand Distance Vector) Routing Algorithm

Routing under the AODV protocol is accomplished using tables in each node that store the next hop (intermediary node between source and destination nodes) for a destination node. If a next hop is not known, route discovery must take place in order to find a path. Since only a limited number of routes can be stored on a Router, route discovery will take place more often on a large network with communication between many different nodes.

Node	Destination Address	Next Hop Address
R3	Router 6	Coordinator
C	Router 6	Router 5
R5	Router 6	Router 6

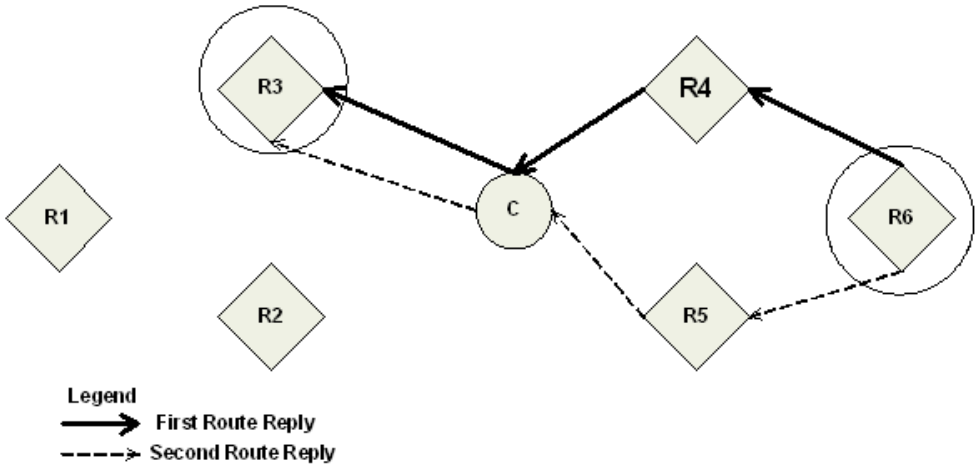
When a source node must discover a route to a destination node, it sends a broadcast route request command. The route request command contains the source network address, the destination network address and a path cost field (a metric for measuring route quality). As the route request command is propagated through the network (refer to the Broadcast Transmission), each node that re-broadcasts the message updates the path cost field and creates a temporary entry in its route discovery table.

Sample Route Request (Broadcast) Transmission Where R3 is Trying to Discover a Route to R6



When the destination node receives a route request, it compares the 'path cost' field against previously received route request commands. If the path cost stored in the route request is better than any previously received, the destination node will transmit a route reply packet to the node that originated the route request. Intermediate nodes receive and forward the route reply packet to the source node (the node that originated route request).

Sample Route Reply (Unicast) Where R6 Sends a Route Reply to R3.



Note: R6 could send multiple replies if it identifies a better route.

Retries and Acknowledgments

ZigBee includes acknowledgment packets at both the Mac and Application Support (APS) layers. When data is transmitted to a remote device, it may traverse multiple hops to reach the destination. As data is transmitted from one node to its neighbor, an acknowledgment packet (Ack) is transmitted in the opposite direction to indicate that the transmission was successfully received. If the Ack is not received, the transmitting device will retransmit the data, up to 4 times. This Ack is called the Mac layer acknowledgment.

In addition, the device that originated the transmission expects to receive an acknowledgment packet (Ack) from the destination device. This Ack will traverse the same path that the data traversed, but in the opposite direction. If the originator fails to receive this Ack, it will retransmit the data, up to 2 times until an Ack is received. This Ack is called the ZigBee APS layer acknowledgment.

Refer to the ZigBee specification for more details.

Many-to-One Routing

In networks where many devices must send data to a central collector or gateway device, AODV mesh routing requires significant overhead. If every device in the network had to discover a route before it could send data to the data collector, the network could easily become inundated with broadcast route discovery messages.

Many-to-one routing is an optimization for these kinds of networks. Rather than require each device to do its own route discovery, a single many-to-one broadcast transmission is sent from the data collector to establish reverse routes on all devices. This is shown in the figure below. The left side shows the many broadcasts the devices can send when they create their own routes and the route replies generated by the data collector. The right side shows the benefits of many-to-one routing where a single broadcast creates reverse routes to the data collector on all routers.

The many-to-one broadcast is a route request message with the target discovery address set to the address of the data collector. Devices that receive this route request create a reverse many-to-one routing table entry to create a path back to the data collector. The ZigBee stack on a device uses historical link quality information about each neighbor to select a reliable neighbor for the reverse route.

When a device sends data to a data collector, and it finds a many-to-one route in its routing table, it will transmit the data without performing a route discovery. The many-to-one route request should be sent periodically to update and refresh the reverse routes in the network.

Applications that require multiple data collectors can also use many-to-one routing. If more than one data collector device sends a many-to-one broadcast, devices will create one reverse routing table entry for each collector.

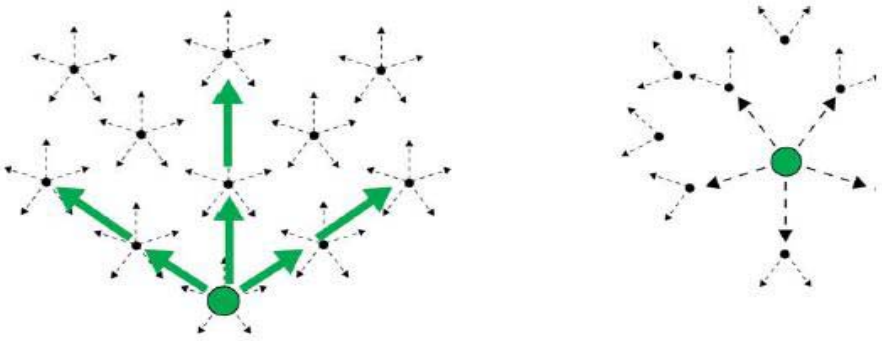
In ZB firmware, the AR command is used to enable many-to-one broadcasting on a device. The AR command sets a time interval (measured in 10 second units) for sending the many to one broadcast transmission. (See the command table for details.)

Source Routing

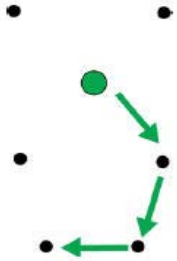
In applications where a device must transmit data to many remotes, AODV routing would require performing one route discovery for each destination device to establish a route. If there are more destination devices than there are routing table entries, established AODV routes would be overwritten with new routes, causing route discoveries to occur more regularly. This could result in larger packet delays and poor network performance.

ZigBee source routing helps solve these problems. In contrast to many-to-one routing that establishes routing paths from many devices to one data collector, source routing allows the collector to store and specify routes for many remotes.

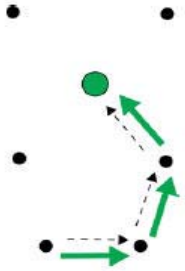
To use source routing, a device must use the API mode, and it must send periodic many-to-one route request broadcasts (AR command) to create a many-to-one route to it on all devices. When remote devices send RF data using a many-to-one route, they first send a route record transmission. The route record transmission is unicast along the many-to-one route until it reaches the data collector. As the route record traverses the many-to-one route, it appends the 16-bit address of each device in the route into the RF payload. When the route record reaches the data collector, it contains the address of the sender, and the 16-bit address of each hop in the route. The data collector can store the routing information and retrieve it later to send a source routed packet to the remote. This is shown in the images below.



The data collector sends a Many-to-One route request broadcast to create reverse routes on all devices.



A remote device sends an RF data packet to the data collector. (This is prefaced by a route record transmission to the data collector.)



After obtaining a source route, the data collector sends a source routed transmission to the remote device.

Acquiring Source Routes

Acquiring source routes requires the remote devices to send a unicast to a data collector (device that sends many-to-one route request broadcasts). There are several ways to force remotes to send route record transmissions.

1. If the application on remote devices periodically sends data to the data collector, each transmission will force a route record to occur.

2. The data collector can issue a network discovery command (ND command) to force all XBee devices to send a network discovery response. Each network discovery response will be prefaced by a route record.

3. Periodic IO sampling can be enabled on remotes to force them to send data at a regular rate. Each IO sample would be prefaced by a route record. (See chapter 8 for details.)

4. If the NI string of the remote device is known, the DN command can be issued with the NI string of the remote in the payload. The remote device with a matching NI string would send a route record and a DN response.

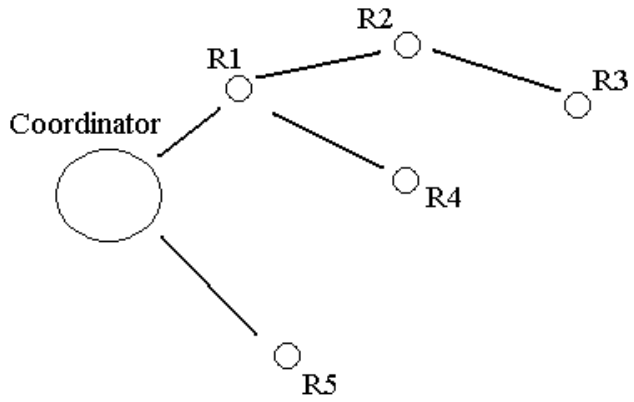
Storing Source Routes

When a data collector receives a route record, it sends it out the serial port as a Route Record Indicator API frame (0xA1). To use source routing, the application should receive these frames and store the source route information.

Sending a Source Routed Transmission

To send a source routed transmission, the application should send a Create Source Route API frame (0x21) to the XBee to create a source route in its internal source route table. After sending the Create Source Route API frame, the application can send data transmission or remote command request frames as needed to the same destination, or any destination in the source route. Once data must be sent to a new destination (a destination not included in the last source route), the application should first send a new Create Source Route API frame. The XBee can buffer one source route that includes up to 11 hops (excluding source and destination).

For example, suppose a network exists with a coordinator and 5 routers (R1, R2, R3, R4, R5) with known source routes as shown below.



To send a source-routed packet to R3, the application must send a Create Source Route API frame (0x21) to the XBee, with a destination of R3, and 2 hops (R1 and R2). If the 64- bit address of R3 is 0x0013A200 404a1234 and the 16-bit addresses of R1, R2, and R3 are:

Device	16-bit address
R1	0xAABB
R2	0xCCDD
R3	0xEEFF

Then the Create Source Route API frame would be:

7E 0012 21 00 0013A200 404A1234 EEFF 00 02 CCDD AABB 5C

Where:

0x0012 - length

0x21 - API ID (create source route)

0x00 - frame ID (set to 0 always)

0x0013A200 404A1234 - 64-bit address of R3 (destination)

0xEEFF - 16-bit address of R3 (destination)

0x00 - Route options (set to 0)

0x02 - Number of intermediate devices in the source route

0xCCDD - Address of furthest device (1-hop from target)

0xAABB - Address of next-closer device

0x5C - Checksum (0xFF - SUM (all bytes after length))

Repairing Source Routes

It is possible in a network to have an existing source route fail (i.e. a device in the route moves or goes down, etc.). If a device goes down in a source routed network, all routes that used the device will be broken.

As mentioned previously, source routing must be used with many-to-one routing. (A device that uses source routing must also send a periodic many-to-one broadcast in order to keep routes fresh). If a source route is broken, remote devices must send in new route record transmissions to the data collector to provide it with a new source route. This requires that remote devices periodically send data transmissions into the data collector. See the earlier "Acquiring Source Routes" section for details.

Retries and Acknowledgments

ZigBee includes acknowledgment packets at both the Mac and Application Support (APS) layers. When data is transmitted to a remote device, it may traverse multiple hops to reach the destination. As data is transmitted from one node to its neighbor, an acknowledgment packet (Ack) is transmitted in the opposite direction to indicate that the transmission was successfully received. If the Ack is not received, the transmitting device will retransmit the data, up to 4 times. This Ack is called the Mac layer acknowledgment.

In addition, the device that originated the transmission expects to receive an acknowledgment packet (Ack) from the destination device. This Ack will traverse the same path that the data traversed, but in the opposite direction. If the originator fails to receive this Ack, it will retransmit the data, up to 2 times until an Ack is received. This Ack is called the ZigBee APS layer acknowledgment.

Refer to the ZigBee specification for more details.

Encrypted Transmissions

Encrypted transmissions are routed similar to non-encrypted transmissions with one exception. As an encrypted packet propagates from one device to another, each device decrypts the packet using the network key, and authenticates the packet by verifying packet integrity. It then re-encrypts the packet with its own source address and frame counter values, and sends the message to the next hop. This process adds some overhead latency to unicast transmissions, but it helps prevent replay attacks. See chapter 5 for details.

Maximum RF Payload Size

XBee ZB firmware includes a command (ATNP) that returns the maximum number of RF payload bytes that can be sent in a unicast transmission. Querying the NP command, like most other commands, returns a hexadecimal value. This number will change based on whether security is enabled or not. If security is enabled (EE command), the maximum number of RF payload bytes decreases since security requires additional overhead.

After reading the NP value, the following conditions can affect the maximum number of data bytes in a single RF transmission:

- Broadcast transmissions can support 8 bytes more than unicast transmissions.
- If source routing is used, the 16-bit addresses in the source route are inserted into the RF payload space. For example, if NP returns 84 bytes, and a source route must traverse 3 intermediate hops (3 16-bit addresses), the total number of bytes that can be sent in one RF packet is 78.
- Enabling APS encryption (API tx option bit set) will reduce the number of payload bytes by 4.

Throughput

Throughput in a ZigBee network can vary by a number of variables, including: number of hops, encryption enabled/disabled, sleeping end devices, failures/route discoveries. Our empirical testing showed the following throughput performance in a robust operating environment (low interference).

Data Throughput*

Configuration	Data Throughput
1 hop, RR, SD	58kbps
1 hop, RR, SE	34kbps
1 hop, RE, SD	Not yet available
1 hop, RE, SE	Not yet available
1 hop, ER, SD	Not yet available
1 hop, ER, SE	Not yet available
4 hops, RR, SD	Not yet available
4 hops, RR, SE	Not yet available

RR = router to router,

RE = router to end device (non-sleeping),

ER = end device (non-sleeping) to router,

SD = security disabled,

SE = security enabled.

4 hops = 5 nodes total, 3 intermediate router nodes

* Data throughput measurements were made setting the serial interface rate to 115200 bps, and measuring the time to send 100,000 bytes from source to destination. During the test, no route discoveries or failures occurred.

Latency Timing Specifications

Timing Specifications

Network Depth	100 Node Network	200 Node Network
1	1-byte packet: 32-byte packet:	1-byte packet: 32-byte packet:
2	1-byte packet: 32-byte packet:	1-byte packet: 32-byte packet:
4	1-byte packet: 32-byte packet:	1-byte packet: 32-byte packet:

ZDO Transmissions

ZigBee defines a ZigBee Device Objects layer (ZDO) that can provide device and service discovery and network management capabilities. This layer is described below.

ZigBee Device Objects (ZDO)

The ZigBee Device Objects (ZDO) is supported to some extent on all ZigBee devices. The ZDO is an endpoint that implements services described in the ZigBee Device Profile in the ZigBee specification. Each service has an assigned cluster ID, and most service requests have an associated response. The following table describes some common ZDO services.

Cluster Name	Cluster ID	Description
Network Address Request	0x0000	Request a 16-bit address of the radio with a matching 64-bit address (required parameter).
Active Endpoints Request	0x0005	Request a list of endpoints from a remote device.

LQI Request	0x0031	Request data from a neighbor table of a remote device.
Routing Table Request	0x0032	Request to retrieve routing table entries from a remote device.
Network Address Response	0x8000	Response that includes the 16-bit address of a device.
LQI Response	0x8031	Response that includes neighbor table data from a remote device.
Routing Table Response	0x8032	Response that includes routing table entry data from a remote device.

Refer to the ZigBee specification for a detailed description of all ZigBee Device Profile services.

Sending a ZDO Command

To send a ZDO command, an explicit transmit API frame must be used and formatted correctly. The source and destination endpoints must be set to 0, and the profile ID must be set to 0. The cluster ID must be set to match the cluster ID of the appropriate service. For example, to send an active endpoints request, the cluster ID must be set to 0x0005.

The first byte of payload in the API frame is an application sequence number (transaction sequence number) that can be set to any single byte value. This same value will be used in the first byte of the ZDO response. All remaining payload bytes must be set as required by the ZDO. All multi-byte values must be sent in little endian byte order.

Receiving ZDO Commands and Responses

In XBee ZB firmware, ZDO commands can easily be sent using the API. In order to receive incoming ZDO commands, receiver application addressing must be enabled with the AO command. (See examples later in this section.) Not all incoming ZDO commands are passed up to the application.

When a ZDO message is received on endpoint 0 and profile ID 0, the cluster ID indicates the type of ZDO message that was received. The first byte of payload is generally a sequence number that corresponds to a sequence number of a request. The remaining bytes are set as defined by the ZDO. Similar to a ZDO request, all multi-byte values in the response are in little endian byte order.

Example 1: Send a ZDO LQI Request to read the neighbor table contents of a remote.

Looking at the ZigBee specification, the cluster ID for an LQI Request is 0x0031, and the payload only requires a single byte (start index). This example will send an LQI request to a remote device with a 64-bit address of 0x0013A200 40401234. The start index will be set to 0, and the transaction sequence number will be set to 0x76

API Frame

7E 0016 11 01 0013A200 40401234 FFFE 00 00 0031 0000 00 00 76 00 CE

0x0016 - length

0x11 - Explicit transmit request

0x01 - frame ID (set to a non-zero value to enable the transmit status message, or set to 0 to disable)

0x0013A200 40401234 - 64-bit address of the remote

0xFFFE - 16-bit address of the remote (0xFFFE = unknown). Optionally, set to the 16-bit address of the destination if known.

0x00 - Source endpoint

0x00 - Destination endpoint

0x0031 - Cluster ID (LQI Request, or Neighbor table request)

0x0000 - Profile ID (ZigBee Device Profile)

0x00 - Broadcast radius

0x00 - Tx Options

0x76 - Transaction sequence number

0x00 - Required payload for LQI request command

0xCE - Checksum (0xFF - SUM (all bytes after length))

Description

This API frame sends a ZDO LQI request (neighbor table request) to a remote device to obtain data from its neighbor table. Recall that the AO command must be set correctly on an API device to enable the explicit API receive frames in order to receive the ZDO response.

Example 2: Send a ZDO Network Address Request to discover the 16-bit address of a remote.

Looking at the ZigBee specification, the cluster ID for a network Address Request is 0x0000, and the payload only requires the following:

[64-bit address] + [Request Type] + [Start Index]

This example will send a Network Address Request as a broadcast transmission to discover the 16-bit address of the device with a 64-bit address of 0x0013A200 40401234. The request type and start index will be set to 0, and the transaction sequence number will be set to 0x44

API Frame

7E 001F 11 01 00000000 0000FFFF FFFE 00 00 0000 0000 00 00 44 34124040 00A21300 00 00 33

0x001F - length

0x11 - Explicit transmit request

0x01 - frame ID (set to a non-zero value to enable the transmit status message, or set to 0 to disable)

0x00000000 0000FFFF - 64-bit address for a broadcast transmission

0xFFFE - Set to this value for a broadcast transmission.

0x00 - Source endpoint

0x00 - Destination endpoint

0x0000 - Cluster ID (Network Address Request)

0x0000 - Profile ID (ZigBee Device Profile)

0x00 - Broadcast radius

0x00 - Tx Options

0x44 - Transaction sequence number

0x34124040 00A21300 00 00 - Required payload for Network Address Request command

0x33 - Checksum (0xFF - SUM (all bytes after length))

Description

This API frame sends a broadcast ZDO Network Address Request to obtain the 16-bit address of a device with a 64-bit address of 0x0013A200 40401234. Note the bytes for the 64-bit address were inserted in little endian byte order. All multi-byte fields in the API payload of a ZDO command must have their data inserted in little endian byte order. Also recall that the AO command must be set correctly on an API device to enable the explicit API receive frames in order to receive the ZDO response.

Transmission Timeouts

The ZigBee stack includes two kinds of transmission timeouts, depending on the nature of the destination device. For destination devices such as routers whose receiver is always on, a unicast timeout is used. The unicast timeout estimates a timeout based on the number of unicast hops the packet should traverse to get data to the destination device. For transmissions destined for end devices, the ZigBee stack uses an extended timeout that includes the unicast timeout (to route data to the end device's parent), and it includes a timeout for the end device to finish sleeping, wake, and poll the parent for data.

The ZigBee stack includes some provisions for a device to detect if the destination is an end device or not. The ZigBee stack uses the unicast timeout unless it knows the destination is an end device.

The XBee API includes a transmit options bit that can be set to specify if the extended timeout should be used for a given transmission. If this bit is set, the extended timeout will be used when sending RF data to the specified destination. To improve routing reliability, applications should set the extended timeout bit when sending data to end devices if:

- The application sends data to 10 or more remote devices, some of which are end devices, AND
- The end devices may sleep longer than the unicast timeout

Equations for these timeouts are computed in the following sections.

Note: The timeouts in this section are worst-case timeouts and should be padded by a few hundred milliseconds. These worst-case timeouts apply when an existing route breaks down (e.g. intermediate hop or destination device moved).

Unicast Timeout

The unicast timeout is settable with the NH command. The actual unicast timeout is computed as ((50 * NH) + 100). The default NH value is 30 which equates to a 1.6 second timeout.

The unicast timeout includes 3 transmission attempts (1 attempt and 2 retries). The maximum total timeout is about:

3 * ((50 * NH) + 100).

For example, if NH=30 (0x1E), the unicast timeout is about

3 * ((50 * 30) + 100), or

3 * (1500 + 100), or

3 * (1600), or

4800 ms, or

4.8 seconds.

Extended Timeout

The worst-case transmission timeout when sending data to an end device is somewhat larger than when transmitting to a router or coordinator. As described later in chapter 6, RF data packets are actually sent to the parent of the end device, who buffers the packet until the end device wakes to receive it. The parent will buffer an RF data packet for up to (1.2 * SP) time.

To ensure the end device has adequate time to wake and receive the data, the extended transmission timeout to an end device is:

(50 * NH) + (1.2 * SP)

This timeout includes the packet buffering timeout (1.2 * SP) and time to account for routing through the mesh network (50 * NH).

If an acknowledgment is not received within this time, the sender will resend the transmission up to two more times. With retries included, the longest transmission timeout when sending data to an end device is:

3 * ((50 * NH) + (1.2 * SP))

The SP value in both equations must be entered in millisecond units. (The SP command setting uses 10ms units and must be converted to milliseconds to be used in this equation.)

For example, suppose a router is configured with NH=30 (0x1E) and SP=0x3E8 (10,000 ms), and that it is either trying to send data to one of its end device children, or to a remote end device. The total extended timeout to the end device is about:

3 * ((50 * NH) + (1.2 * SP)), or

3 * (1500 + 12000), or

3 * (13500), or

40500 ms, or

40.5 seconds.

Transmission Examples

Example 1: Send a unicast API data transmission to the coordinator using 64-bit address 0, with payload "TxData".

API Frame

7E 0014 10 01 00000000 00000000 FFFE 00 00 54 78 44 61 74 61 AB

Field Composition

0x0014 - length

0x10 - API ID (tx data)

0x01 - frame ID (set greater than 0 to enable the tx-status response)

0x00000000 00000000 - 64-bit address of coordinator (ZB definition)

0xFFFE - Required 16-bit address if sending data to 64-bit address of 0.

0x00 - Broadcast radius (0 = max hops)

0x00 - Tx options

0x54 78 44 61 74 61 - ASCII representation of "TxData" string

0xAB - Checksum (0xFF - SUM (all bytes after length))

Description

This transmission sends the string "TxData" to the coordinator, without knowing the coordinator device's 64-bit address. A 64-bit address of 0 is defined as the coordinator in ZB firmware. If the coordinator's 64-bit address was known, the 64-bit address of 0 could be replaced with the coordinator's 64-bit address, and the 16-bit address could be set to 0.

Example 2 - Send a broadcast API data transmission that all devices can receive (including sleeping end devices), with payload "TxData".

API Frame

7E 0014 10 01 00000000 0000FFFF FFFE 00 00 54 78 44 61 74 61 AD

Field Composition

0x0014 - length

0x10 - API ID (tx data)

0x01 - frame ID (set to a non-zero value to enable the tx-status response)

0x00000000 0000FFFF - Broadcast definition (including sleeping end devices

0xFFFE - Required 16-bit address to send broadcast transmission.

0x00 - Broadcast radius (0 = max hops)

0x00 - Tx options

0x54 78 44 61 74 61 - ASCII representation of "TxData" string

0xAD - Checksum (0xFF - SUM (all bytes after length))

Description

This transmission sends the string "TxData" as a broadcast transmission. Since the destination address is set to 0xFFFF, all devices, including sleeping end devices can receive this broadcast.

If receiver application addressing is enabled, the XBee will report all received data frames in the explicit format (0x91) to indicate the source and destination endpoints, cluster ID, and profile ID that each packet was received on. (Status messages like modem status and route record indicators are not affected.)

To enable receiver application addressing, set the AO command to 1 using the AT command frame (0x08). Here's how to do this:

API Frame

7E 0005 08 01 414F 01 65

Field Composition

0x0005 - length

0x08 - API ID (at command)

0x01 - frame ID (set to a non-zero value to enable AT command response frames)

0x414F - ASCII representation of 'A','O' (the command being issued)

0x01 - Parameter value

0x65 - Checksum (0xFF - SUM (all bytes after length))

Description

Setting AO=1 is required for the XBee to use the explicit receive API frame (0x91) when RF data packets are received. This is required if the application needs indication of source or destination endpoint, cluster ID, and/or profile ID values used in received ZigBee data packets. ZDO messages can only be received if AO=1.

5. Security

ZigBee supports various levels of security that can be configured depending on the needs of the application. Security provisions include:

- 128-bit AES encryption
- Two security keys that can be preconfigured or obtained during joining
- Support for a trust center
- Provisions to ensure message integrity, confidentiality, and authentication.

The first half of this chapter describes various security features defined in the ZigBee specification, while the last half illustrates how the XBee modules can be configured to support these features

Security Modes

The ZigBee standard supports three security modes – residential, standard, and high security. Residential security was first supported in the ZigBee 2006 standard. This level of security requires a network key be shared among devices. Standard security adds a number of optional security enhancements over residential security, including an APS layer link key. High security adds entity authentication, and a number of other features not widely supported.

XBee ZB modules primarily support standard security, although end devices that support residential security can join and interoperate with standard security devices. The remainder of this chapter focuses on material that is relevant to standard security.

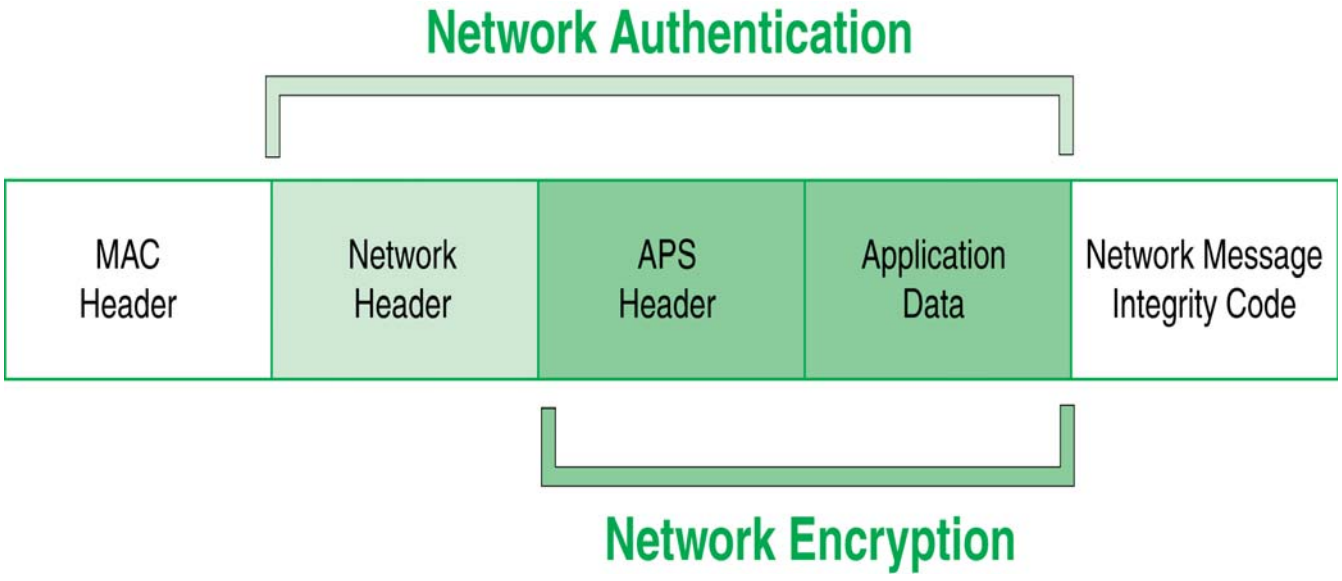
ZigBee Security Model

ZigBee security is applied to the Network and APS layers. Packets are encrypted with 128-bit AES encryption. A network key and optional link key can be used to encrypt data. Only devices with the same keys are able to communicate together in a network. Routers and end devices that will communicate on a secure network must obtain the correct security keys.

Network Layer Security

The network key is used to encrypt the APS layer and application data. In addition to encrypting application messages, network security is also applied to route request and reply messages, APS commands, and ZDO commands. Network encryption is not applied to MAC layer transmissions such as beacon transmissions, etc. If security is enabled in a network, all data packets will be encrypted with the network key.

Packets are encrypted and authenticated using 128-bit AES. This is shown in the figure below.



Frame Counter

The network header of encrypted packets includes a 32-bit frame counter. Each device in the network maintains a 32-bit frame counter that is incremented for every transmission. In addition, devices track the last known 32-bit frame counter for each of its neighbors. If a device receives a packet from a neighbor with a smaller frame counter than it has previously seen, the packet is discarded. The frame counter is used to protect against replay attacks.

If the frame counter reaches a maximum value of 0xFFFFFFFF, it does not wrap to 0 and no more transmissions can be sent. Due to the size of the frame counters, reaching the maximum value is a very unlikely event for most applications. The following table shows the required time under different conditions, for the frame counter to reach its maximum value.

Average Transmission Rate	Time until 32-bit frame counter expires
1 / second	136 years
10 / second	13.6 years

To clear the frame counters without compromising security, the network key can be changed in the network. When the network key is updated, the frame counters on all devices reset to 0. (See the Network Key Updates section for details.)

Message Integrity Code

The network header, APS header, and application data are all authenticated with 128-bit AES. A hash is performed on these fields and is appended as a 4-byte message integrity code (MIC) to the end of the packet. The MIC allows receiving devices to ensure the message has not been changed. The MIC provides message integrity in the ZigBee security model. If a device receives a packet and the MIC does not match the device's own hash of the data, the packet is dropped.

Network Layer Encryption and Decryption

Packets with network layer encryption are encrypted and decrypted by each hop in a route. When a device receives a packet with network encryption, it decrypts the packet and authenticates the packet. If the device is not the destination, it then encrypts and authenticates the packet, using its own frame counter and source address in the network header section.

Since network encryption is performed at each hop, packet latency is slightly longer in an encrypted network than in a non-encrypted network. Also, security requires 18 bytes of overhead to include a 32-bit frame counter, an 8-byte source address, 4-byte MIC, and 2 other bytes. This reduces the number of payload bytes that can be sent in a data packet.

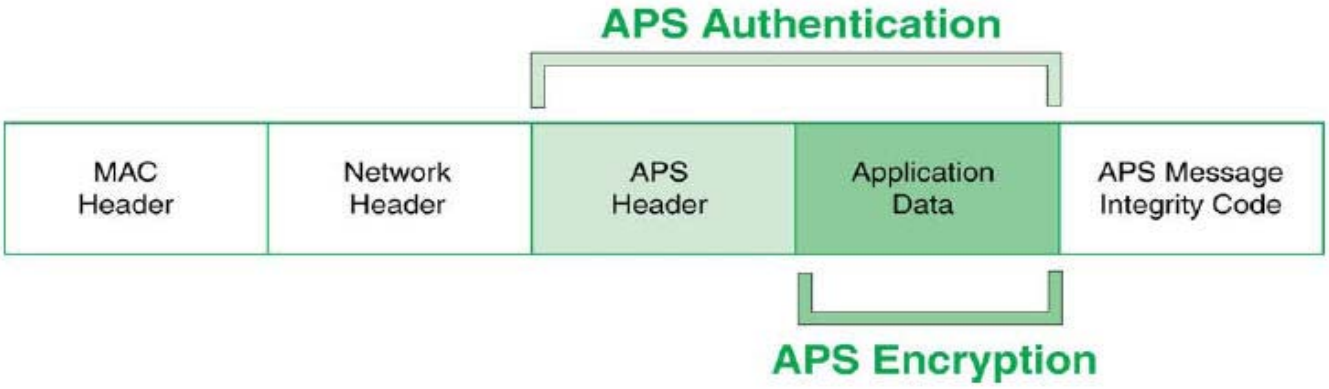
Network Key Updates

ZigBee supports a mechanism for changing the network key in a network. When the network key is changed, the frame counters in all devices reset to 0.

APS Layer Security

APS layer security can be used to encrypt application data using a key that is shared between source and destination devices. Where network layer security is applied to all data transmissions and is decrypted and re-encrypted on a hop-by-hop basis, APS security is optional and provides end-to-end security using an APS link key that only the source and destination device know. APS security can be applied on a packet-by-packet basis. APS security cannot be applied to broadcast transmissions.

If APS security is enabled, packets are encrypted and authenticated using 128-bit AES. This is shown in the figure below:



Message integrity Code

If APS security is enabled, the APS header and data payload are authenticated with 128-bit AES. A hash is performed on these fields and appended as a 4-byte message integrity code (MIC) to the end of the packet. This MIC is different than the MIC appended by the network layer. The MIC allows the destination device to ensure the message has not been changed. If the destination device receives a packet and the MIC does not match the destination device's own hash of the data, the packet is dropped.

APS Link Keys

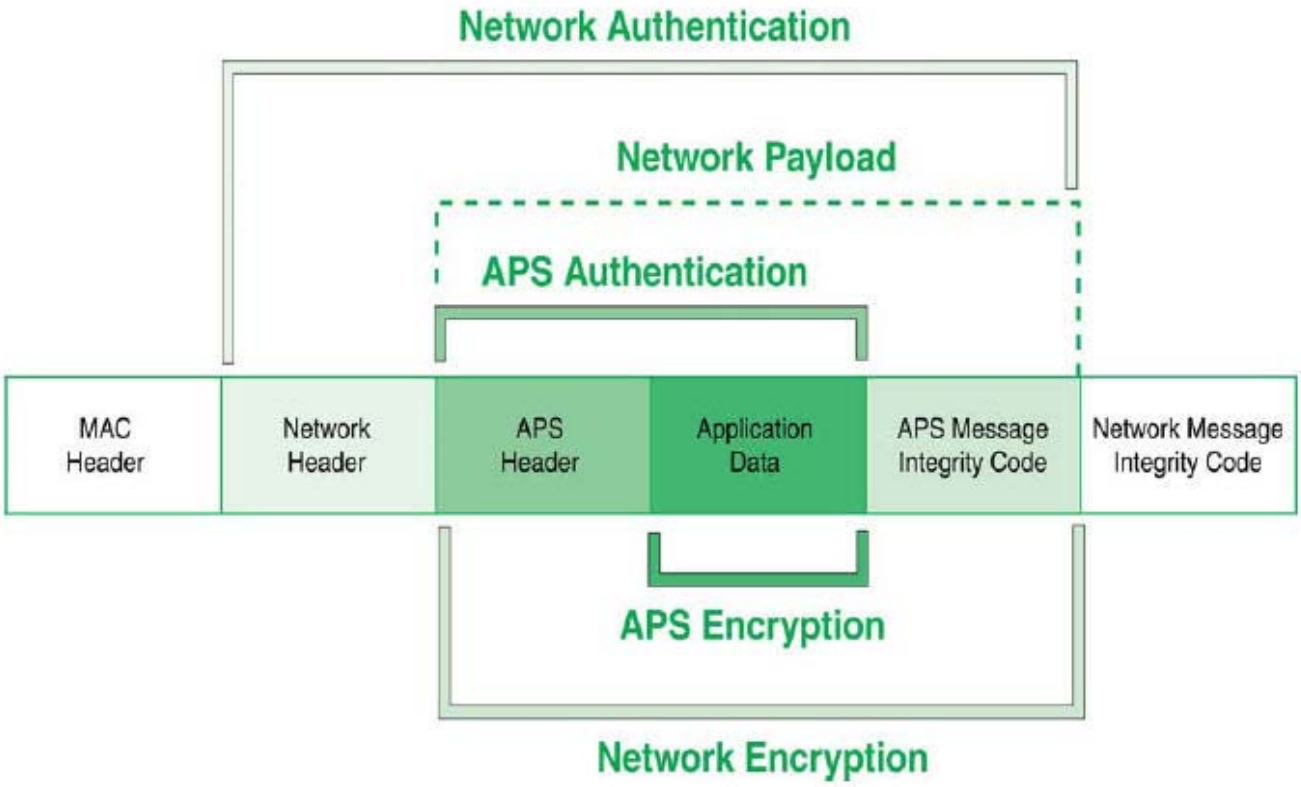
There are two kinds of APS link keys – trust center link keys and application link keys. A trust center link key is established between a device and the trust center, where an application link key is established between a device and another device in the network where neither device is the trust center.

APS Layer Encryption and Decryption

Packets with APS layer encryption are encrypted at the source and only decrypted by the destination. Since APS encryption requires a 5-byte header and a 4-byte MIC, the maximum data payload is reduced by 9 bytes when APS encryption is used.

Network and APS Layer Encryption

Network and APS layer encryption can both be applied to data. The following figure demonstrates the authentication and encryption performed on the final ZigBee packet when both are applied.



Trust Center

ZigBee defines a trust center device that is responsible for authenticating devices that join the network. The trust center also manages link key distribution in the network.

Forming and Joining a Secure Network

The coordinator is responsible for selecting a network encryption key. This key can either be preconfigured or randomly selected. In addition, the coordinator generally operates as a trust center and must therefore select the trust center link key. The trust center link key can also be preconfigured or randomly selected.

Devices that join the network must obtain the network key when they join. When a device joins a secure network, the network and link keys can be sent to the joining device. If the joining device has a pre-configured trust center link key, the network key will be sent to the joining device encrypted by the link key. Otherwise, if the joining device is not pre-configured with the link key, the device could only join the network if the network key is sent unencrypted ("in the clear"). The trust center must decide whether or not to send the network key unencrypted to joining devices that are not pre-configured with the link key. Sending the network key unencrypted is not recommended as it can open a security hole in the network. To maximize security, devices should be pre-configured with the correct link key.

Implementing Security on the XBee

If security is enabled in the XBee ZB firmware, devices acquire the network key when they join a network. Data transmissions are always encrypted with the network key, and can optionally be end-to-end encrypted with the APS link key. The following sections discuss the security settings and options in the XBee ZB firmware.

Enabling Security

To enable security on a device, the EE command must be set to 1. If the EE command value is changed and changes are applied (e.g. AC command), the XBee module will leave the network (PAN ID and channel) it was operating on, and attempt to form or join a new network.

If EE is set to 1, all data transmissions will be encrypted with the network key. When security is enabled, the maximum number of bytes in a single RF transmission will be reduced. See the NP command for details.

Note: The EE command must be set the same on all devices in a network. Changes to the EE command should be written to non-volatile memory (to be preserved through power cycle or reset events) using the WR command.

Setting the Network Security Key

The coordinator must select the network security key for the network. The NK command (write-only) is used to set the network key. If NK=0 (default), a random network key will be selected. (This should suffice for most applications.) Otherwise, if NK is set to a non-zero value, the network security key will use the value specified by NK. NK is only supported on the coordinator.

Routers and end devices with security enabled (ATEE=1) acquire the network key when they join a network. They will receive the network key encrypted with the link key if they share a pre-configured link key with the coordinator. See the following section for details.

Setting the APS Trust Center Link Key

The coordinator must also select the trust center link key, using the KY command. If KY=0 (default), the coordinator will select a random trust center link key (not recommended). Otherwise, if KY is set greater than 0, this value will be used as the pre-configured trust center link key. KY is write-only and cannot be read.

Note: Application link keys (sent between two devices where neither device is the coordinator) are not supported in ZB firmware at this time.

Random Trust Center Link Keys

If the coordinator selects a random trust center link key (KY=0, default), then it will allow devices to join the network without having a pre-configured link key. However, this will cause the network key to be sent unencrypted over-the-air to joining devices and is not recommended.

Pre-configured Trust Center Link Keys

If the coordinator uses a pre-configured link key (KY > 0), then the coordinator will not send the network key unencrypted to joining devices. Only devices with the correct pre-configured link key will be able to join and communicate on the network.

Enabling APS Encryption

APS encryption is an optional layer of security that uses the link key to encrypt the data payload. Unlike network encryption that is decrypted and encrypted on a hop-by-hop basis, APS encryption is only decrypted by the destination device. The XBee must be configured with security enabled (EE set to 1) to use APS encryption.

APS encryption can be enabled in API mode on a per-packet basis. To enable APS encryption for a given transmission, the "enable APS encryption" transmit options bit should be set in the API transmit frame. Enabling APS encryption decreases the maximum payload size by 9 bytes.

Using a Trust Center

The EO command can be used to define the coordinator as a trust center. If the coordinator is a trust center, it will be alerted to all new join attempts in the network. The trust center also has the ability to update or change the network key on the network.

In ZB firmware, a secure network can be established with or without a trust center. Network and APS layer encryption are supported if a trust center is used or not.

Updating the Network Key with a Trust Center

If the trust center has started a network and the NK value is changed, the coordinator will update the network key on all devices in the network. (Changes to NK will not force the device to leave the network.) The network will continue to operate on the same channel and PAN ID, but the devices in the network will update their network key, increment their network key sequence number, and restore their frame counters to 0.

Updating the Network Key without a Trust Center

If the coordinator is not running as a trust center, the network reset command (NR1) can be used to force all devices in the network to leave the current network and rejoin the network on another channel. When devices leave and reform then network, the frame counters are reset to 0. This approach will cause the coordinator to form a new network that the remaining devices should join. Resetting the network in this manner will bring the coordinator and routers in the network down for about 10 seconds, and will likely cause the 16-bit PAN ID and 16-bit addresses of the devices to change.

XBee Security Examples

This section covers some sample XBee configurations to support different security modes. Several AT commands are listed with suggested parameter values. The notation in this section includes an '=' sign to indicate what each command register should be set to - for example, EE=1. This is not the correct notation for setting command values in the XBee. In AT command mode, each command is issued with a leading 'AT' and no '=' sign - for example ATEE1. In the API, the two byte command is used in the command field, and parameters are populated as binary values in the parameter field.

Example 1: Forming a network with security (pre-configured link keys)

1. Start a coordinator with the following settings:
   a. ID=2234 (arbitrarily selected)
   b. EE=1
   c. NK=0
   d. KY=4455
   e. WR (save networking parameters to preserve them through power cycle)
2. Configure one or more routers or end devices with the following settings:
   a. ID=2234
   b. EE=1
   c. KY=4455
   d. WR (save networking parameters to preserve them through power cycle)
3. Read the AI setting on the coordinator and joining devices until they return 0 (formed or joined a network).

In this example, EE, ID, and KY are set the same on all devices. After successfully joining the secure network, all application data transmissions will be encrypted by the network key. Since NK was set to 0 on the coordinator, a random network key was selected. And since the link key (KY) was configured the same on all devices, to a non-zero value, the network key was sent encrypted by the pre-configured link key (KY) when the devices joined.

Example 2: Forming a network with security (obtaining keys during joining)

1. Start a coordinator with the following settings:
   a. ID=2235
   b. EE=1
   c. NK=0
   d. KY=0

e. WR (save networking parameters to preserve them through power cycle)

2. Configure one or more routers or end devices with the following settings:

a. ID=2235

b. EE=1

c. KY=0

d. WR (save networking parameters to preserve them through power cycle)

3. Read the AI setting on the coordinator and joining devices until they return 0 (formed or joined a network).

In this example, EE, ID, and KY are set the same on all devices. Since NK was set to 0 on the coordinator, a random network key was selected. And since KY was set to 0 on all devices, the network key was sent unencrypted ("in the clear") when the devices joined. This approach introduces a security vulnerability into the network and is not recommended.

6. Network Commissioning and Diagnostics

Network commissioning is the process whereby devices in a mesh network are discovered and configured for operation. The XBee modules include several features to support device discovery and configuration. In addition to configuring devices, a strategy must be developed to place devices to ensure reliable routes.

To accommodate these requirements, the XBee modules include various features to aid in device placement, configuration, and network diagnostics.

Device Configuration

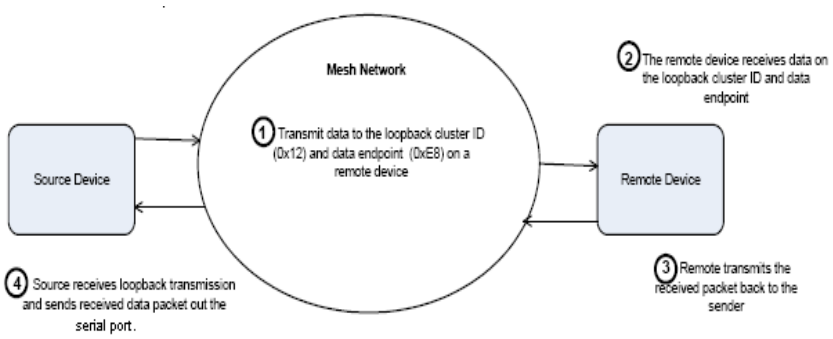
XBee modules can be configured locally through serial commands (AT or API), or remotely through remote API commands. API devices can send configuration commands to set or read the configuration settings of any device in the network.

Device Placement

For a mesh network installation to be successful, the installer must be able to determine where to place individual XBee devices to establish reliable links throughout the mesh network.

Link Testing

A good way to measure the performance of a mesh network is to send unicast data through the network from one device to another to determine the success rate of many transmissions. To simplify link testing, the modules support a loopback cluster ID (0x12) on the data endpoint (0xE8). Any data sent to this cluster ID on the data endpoint will be transmitted back to the sender. This is shown in the figure below:



Demonstration of how the loopback cluster ID and data endpoint can be used to measure the link quality in a mesh network

The configuration steps to send data to the loopback cluster ID depend on the serial port mode as determined by the AP command.

Transparent Mode

To send data to the loopback cluster ID on the data endpoint of a remote device, set the CI command value to 0x12. The SE and DE commands should be set to 0xE8 (default value). The DH and DL commands should be set to the address of the remote (0 for the coordinator, or the 64-bit address of the remote). After exiting command mode, any received serial characters will be transmitted to the remote device, and returned to the sender.

API Mode

Send an Explicit Addressing ZigBee Command API frame (0x11) using 0x12 as the cluster ID and 0xE8 as the source and destination endpoint. Data packets received by the remote will be echoed back to the sender.

RSSI Indicators

It is possible to measure the received signal strength on a device using the DB command. DB returns the RSSI value (measured in –dBm) of the last received packet. However, this number can be misleading. The DB value only indicates the received signal strength of the last hop. If a transmission spans multiple hops, the DB value provides no indication of the overall transmission path, or the quality of the worst link – it only indicates the quality of the last link and should be used sparingly.

The DB value can be determined in hardware using the RSSI/PWM module pin (pin 6). If the RSSI PWM functionality is enabled (P0 command), when the module receives data, the RSSI PWM is set to a value based on the RSSI of the received packet. (Again, this value only indicates the quality of the last hop.) This pin could potentially be connected to an LED to indicate if the link is stable or not.

Device Discovery

Network Discovery

The network discovery command can be used to discover all Digi modules that have joined a network. Issuing the ND command sends a broadcast node discovery command throughout the network. All devices that receive the command will send a response that includes the device's addressing information, node identifier string (see NI command), and other relevant information. This command is useful for generating a list of all module addresses in a network.

When a device receives the node discovery command, it waits a random time before sending its own response. The maximum time delay is set on the ND sender with the NT command. The ND originator includes its NT setting in the transmission to provide a delay window for all devices in the network. Large networks may need to increase NT to improve network discovery reliability. The default NT value is 0x3C (6 seconds).

ZDO Discovery

The ZigBee Device Profile includes provisions to discover devices in a network that are supported on all ZigBee devices (including non-Digi products). These include the LQI Request (cluster ID 0x0031) and the Network Update Request (cluster ID 0x0038). The LQI Request can be used to read the devices in the neighbor table of a remote device, and the Network Update Request can be used to have a remote device do an active scan to discover all nearby ZigBee devices. Both of these ZDO commands can be sent using the XBee Explicit API transmit frame (0x11). See the API chapter for details. Refer to the ZigBee specification for formatting details of these two ZDO frames.

Joining Announce

All ZigBee devices send a ZDO Device Announce broadcast transmission when they join a ZigBee network (ZDO cluster ID 0x0013). These frames will be sent out the XBee's serial port as an Explicit Rx Indicator API frame (0x91) if AO is set to 1. The device announce payload includes the following information:

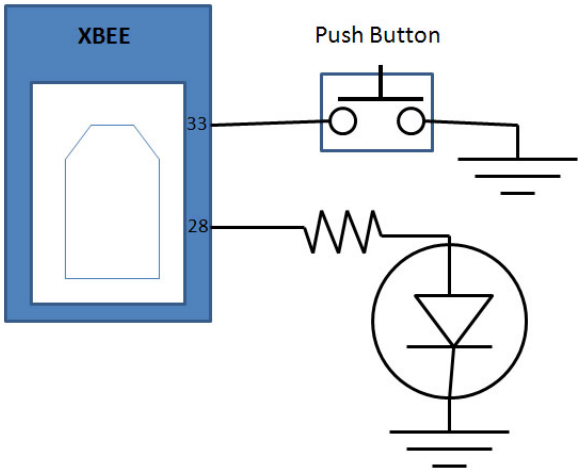
[Sequence Number] + [16-bit address] + [64-bit address] + [Capability]

The 16-bit and 64-bit addresses are received in little-endian byte order (LSB first). See the ZigBee specification for details.

Commissioning Pushbutton and Associate LED

The XBee modules support a set of commissioning and LED behaviors to aid in device deployment and commissioning. These include the commissioning pushbutton definitions and associate LED behaviors. These features can be supported in hardware as shown below.

Commissioning Pushbutton and Associate LED Functionalities



A pushbutton and an LED can be connected to module pins 33 and 28 respectively to support the commisioning pushbutton and Associate LED functionalities.

Commissioning Pushbutton

The commissioning pushbutton definitions provide a variety of simple functions to aid in deploying devices in a network. The commissioning button functionality on pin 33 is enabled by setting the D0 command to 1 (enabled by default).

Button Presses	If module is joined to a network	If module is not joined to a network
1	• Wakes an end device for 30 seconds • Sends a node identification broadcast transmission	• Wakes an end device for 30 seconds • Blinks a numeric error code on the Associate pin indicating the cause of join failure (see section 6.4.2).
2	• Sends a broadcast transmission to enable joining on the coordinator and all devices in the network for 1 minute. (If joining is permanently enabled on a device (NJ = 0xFF), this action has no effect on that device.)	• N/A
4	• Causes the device to leave the PAN. • Issues ATRE to restore module parameters to default values, including ID and SC. • The device attempts to join a network based on its ID and SC settings.	• Issues ATRE to restore module parameters to default values, including ID and SC. • The device attempts to join a network based on its ID and SC settings.

Button presses may be simulated in software using the ATCB command. ATCB should be issued with a parameter set to the number of button presses to execute. (e.g. sending ATCB1 will execute the action(s) associated with a single button press.)

The node identification frame is similar to the node discovery response frame – it contains the device's address, node identifier string (NI command), and other relevant data. All API devices that receive the node identification frame send it out their serial port as an API Node Identification Indicator frame (0x95).

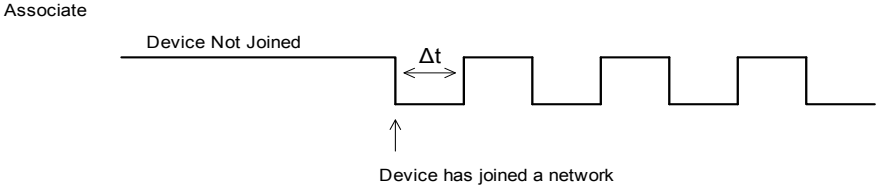
Associate LED

The Associate pin (pin 28) can provide indication of the device's network status and diagnostics information. To take advantage of these indications, an LED can be connected to the Associate pin as shown in the figure above. The Associate LED functionality is enabled by setting the D5 command to 1 (enabled by default). If enabled, the Associate pin is configured as an output and will behave as described in the following sections.

Joined Indication

The Associate pin indicates the network status of a device. If the module is not joined to a network, the Associate pin is set high. Once the module successfully joins a network, the Associate pin blinks at a regular time interval. This is shown in the following figure.

Joined Status of a Device



The associate pin can indicate the joined status of a device . Once the device has joined a network, the associate pin toggles state at a regular interval (Δt). The time can be set by using the LT command.

The LT command defines the blink time of the Associate pin. If set to 0, the device uses the default blink time (500ms for coordinator, 250ms for routers and end devices).

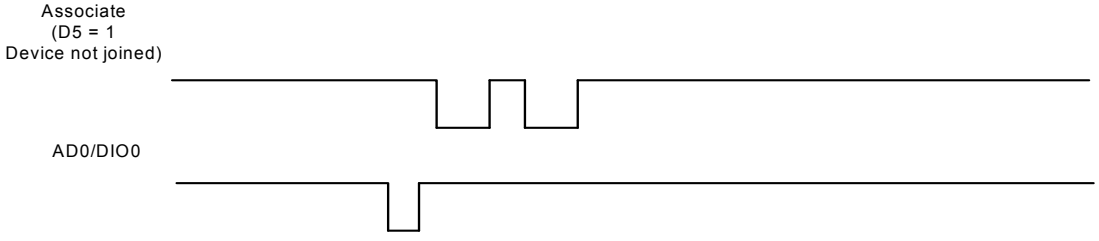
Diagnostics Support

The Associate pin works with the commissioning pushbutton to provide additional diagnostics behaviors to aid in deploying and testing a network. If the commissioning push button is pressed once, and the device has not joined a network, the Associate pin blinks a numeric error code to indicate the cause of join failure. The number of blinks is equal to (AI value – 0x20). For example, if AI=0x22, 2 blinks occur.

If the commissioning push button is pressed once, and the device has joined a network, the device transmits a broadcast node identification packet. If the Associate LED functionality is enabled (D5 command), a device that receives this transmission will blink its Associate pin rapidly for 1 second.

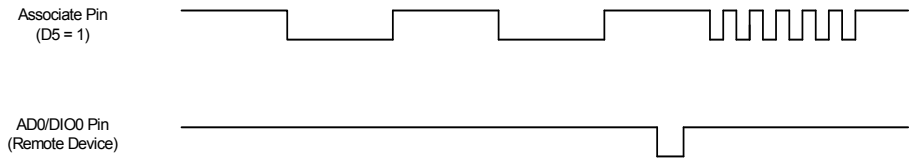
The following figures demonstrate these behaviors.

AI = 0x22



A single commissioning button press when the device has not joined a network that causes the associate pin to blink to indicate the AI Code where: AI = # blinks + 0x20. In this example, AI = 0x22.

Broadcast Node Identification Transmission



A single button press on a remote device causes a broadcast node identification transmission to be sent. All devices that receive this transmission blink their associate pin rapidly for one second if the associate LED functionality is enabled. (D5 = 1)

Binding

There are three binding request messages supported by the Digi XBee firmware: End Device Bind, Bind, and Unbind.

End_Device_Bind_req

The End Device Bind request (ZDO cluster 0x0020) is described in the ZigBee Specification in section 2.4.3.2.1.

During a deployment, an installer may need to bind a switch to a light. He presses a commissioning button sequence on each device. This causes them to send End_Device_Bind_req messages to the Coordinator within a time window (60 s). The payload of each message is a simple descriptor which lists input and output clusterIDs. The Coordinator matches the requests by pairing complementary clusterIDs. After a match has been made, it sends messages to bind the devices together. When the process is over, both devices will have entries in their binding tables which support indirect addressing of messages between their bound endpoints.

R1->C End_Device_Bind_req

R2->C End_Device_Bind_req

R1, R2 send End_Device_Bind_req within 60 s of each other to C

C matches the requests.

C tests one to see if binding is already in place:

R2<-C Unbind_req

R2->C Unbind-rsp (status code - NO_ENTRY)

C proceeds to create binding table entries on the two devices.

R1<-C Bind_req

R1->C Bind_rsp

R2<-C Bind_req

R2->C Bind_rsp

C sends responses to the original End_Device_Bind_req messages.

R1-<C End_Device_Bind_rsp

R2-<C End_Device_Bind_rsp

End Device Binding Sequence (Binding)

This message has a toggle action. If the same two devices were to subsequently send End_Device_Bind_req messages to the Coordinator, the Coordinator would detect they were already bound, and then send Unbind_req messages to remove the binding.

An installer can use this to remove a binding which was made incorrectly, say from a switch to the wrong lamp, simply by repeating the commissioning button sequence he used beforehand.

R1->C End_Device_Bind_req

R2->C End_Device_Bind_req

R1, R2 send End_Device_Bind_req within 60 s of each other to C

C matches the requests.

C tests one to see if binding is already in place:

R2<-C Unbind_req

R2->C Unbind-rsp (status code - SUCCESS)

C proceeds to remove binding table entries from the two devices.

R1<-C Unbind_req

R1->C Unbind_rsp

R2<-C Unbind_req

R2->C Unbind_rsp

C sends responses to the original End_Device_Bind_req messages.

R1-<C End_Device_Bind_rsp

R2-<C End_Device_Bind_rsp

End Device Binding Sequence (Removal)

Here is an example of a correctly formatted End_Device_Bind_req (ZDO cluster 0x0020) using a Digi 0x11 Explicit API Frame:

The frame as a bytelist:

7e00281101000000000000000fffe0000002000000000001f2995cb5474000a21300e605c10101000102004
6

Same frame broken into labeled fields. Note the multibyte fields are represented in big-endian format.

7e	Frame Delimiter
0028	Frame Length
11	API Frame Type (Explicit Frame)
01	Frame Identifier (for response matching)
0000000000000000	Coordinator address
fffe	Code for unknown network address
00	Source Endpoint (need not be 0x00)
00	Destination Endpoint (ZDO endpoint)
0020	Cluster 0x0020 (End_Device_Bind_req)
0000	ProfileID (ZDO)
00	Radius (default, maximum hops)
00	Transmit Options

01f2995cb5474000a21300e605c1010100010200RFData (ZDO payload)

46 Checksum

Here is the RFData (the ZDO payload) broken into labeled fields. Note the multibyte fields of a ZDO payload are represented in little-endian format.

01 Transaction Sequence Number

f299 Binding Target (16 bit network address of sending device)

5cb5474000a21300 (64 bit address of sending device)

e6 Source Endpoint on sending device

05c1 ProfileID (0xC105) - used when matching End_Device_Bind_requests

01 Number of input clusters

0100 Input cluster ID list (0x0100)

01 Number of output clusters

0200 Output cluster ID list (0x0200)

Example of a End_Device_Bind_req

Bind_req

The Bind request (ZDO cluster 0x0021) is described in the ZigBee Specification in section 2.4.3.2.2.

A binding may be coded for either a unicast or a multicast/groupID message.

Unbind_req

The Unbind request (ZDO cluster 0x0022) is described in the ZigBee Specification in section 2.4.3.2.3.

Group Table API

Unlike the Binding Table which is managed with ZDO commands, a ZigBee Group Table is managed by the ZigBee Cluster Library Groups Cluster (0x0006) with ZCL commands.

The Digi ZigBee XBee firmware is intended to work with an external processor where a Public Application Profile with endpoints and clusters is implemented, including a Groups Cluster. The ZigBee XBee firmware should be configured to forward all ZCL commands addressed to this Group Cluster out the UART (see ATAO3). The ZigBee XBee will not use remote Groups Cluster commands to manage its own Group Table.

But for the sake of implementing multicast (group) addressing within the XBee, the external processor must keep the XBee's group table state in synch with its own. And so a Group Table API has been defined whereby the external processor can manage the state of the ZigBee XBee's group table.

The design of the Group Table API of the XBee firmware is derived from the ZCL Group Cluster 0x0006. Developers should use the Explicit API frame 0x11 addressed to the Digi Device Object endpoint (0xE6) with the Digi XBee ProfileID (0xC105) to send commands and requests to the local device.

As a design note, the Home Automation public application profile (section 5.9.5) says groups should only be used for sets of more than 5 devices. This implies sets of 5 or fewer devices should be managed with multiple binding table entries.

There are five commands implemented in the API: Add Group, View Group, Get Group Membership, Remove Group, and Remove All Groups.

There is a sixth command of the Group Cluster described in the ZCL, Add Group If Identifying, which is not supported in this API, because its implementation requires access to the Identify Cluster, which is not maintained on the XBee. The external processor will need to implement that server command while using the Group Table API to keep the XBee's group table in synch using the five command primitives described hereafter.

Add Group

The purpose of the Add Group command is to add a group table entry to associate an active endpoint with a groupID and optionally a groupName. The groupID is a two byte value. The groupName consists of zero to 16 ASCII characters.

The intent of the example which follows is to add a group table entry which associates endpoint E7 with groupID 1234 and groupName "ABCD".

The example packet is given in three parts, the preamble, ZCL Header, and ZCL payload:

Preamble = "11 01 "+LocalDevice64Addr+"FFFE E6 E7 0006 C105 00 00"

The packet is addressed to the local node, using a source endpoint of 0xE6, clusterID of 0x0006, and profileID of 0xC105. The destination endpoint E7 holds the endpoint parameter for the "Add Group" command.

ZCL_header = "01 ee 00"

The first field (byte) is a frame control field which specifies a Cluster Specific command (0x01) using a Client->Server direction(0x00). The second field is a transaction sequence number which is used to associate the response with the command request. The third field is the command identifier for "Add Group" (0x00)[2].

ZCL_payload = "3412 04 41 42 43 44"

The first two bytes is the group Id to add in little endian representation. The next byte is the string name length (00 if no string is wanted). The other bytes are the descriptive ASCII string name ("ABCD") for the group table entry. Note the string is represented with its length in the first byte, and the other bytes containing the ASCII characters.

The example packet in raw hex byte form:

7e001e11010013a2004047b55cfffee6e70006c105000001ee0034120441424344c7

The response in raw hex byte form, consisting of two packets:

7e0018910013a2004047b55cfffee7e68006c1050009ee0000341238

7e00078b01fffe00000076

The response in decoded form:

ZigBee Explicit Rx Indicator

API 0x91 64DestAddr 0x0013A2004047B55C 16DestAddr 0xFFFE SrcEP 0xE7 DestEP 0xE6

ClusterID 0x8006 ProfileID 0xC105 Options 0x00

RF_Data 0x09EE00003412

The response in terms of Preamble, ZCL Header, and ZCL payload:

Preamble = "910013a2004047b55cfffee7e68006c10500"

The packet has its endpoint values reversed from the request, and the clusterID is 0x8006 indicating a Group cluster response.

ZCL_header = "09 ee 00"

The first field is a frame control field which specifies a Cluster Specific command (0x01) using a Server-> Client direction. The second field is a transaction sequence number which is used to associate the response with the command request. The third field is the command identifier "Add Group" (0x00)[2].

ZCL_payload = "00 3412"

The first byte is a status byte (SUCCESS=0x00)[3][4]. The next two bytes hold the group ID (0x1234) in little endian form.

And here is the decoded second message, which is a Tx Status for the original command request. If the FrameId value in the original command request had been zero, or if no space was available in the transmit UART buffer, then no Tx Status message would occur.

ZigBee Tx Status

API 0x8B FrameID 0x01 16DestAddr 0xFFFE

Transmit Retries 0x00 Delivery Status 0x00 Discovery Status 0x00 Success

View Group

The purpose of the View Group command is to get the name string which is associated with a particular endpoint and groupID.

The intent of the example is to get the name string associated with the endpoint E7 and groupID 1234.

The packet:

Preamble = "11 01 "+LocalDevice64Addr+"FFFE E6 E7 0006 C105 00 00"

The packet is addressed to the local node, using a source endpoint of 0xE6, clusterID of 0x0006, and profileID of 0xC105. The destination endpoint E7 is the endpoint parameter for the "View Group" command.

ZCL_header = "01 ee 01"

The first field is a frame control field which specifies a Cluster Specific command (0x01) using a Client->Server direction(0x00). The second field is a transaction sequence number which is used to associate the response with the command request. The third field is the command identifier "View Group" (0x01) [5].

ZCL_payload = "3412"

The two byte value is the groupID in little-endian representation.

The packet in raw hex byte form:

7e001911010013a2004047b55cfffee6e70006c105000001ee013412d4

The response in raw hex byte form, consisting of two packets:

7e001d910013a2004047b55cfffee7e68006c1050009ee01003412044142434424

7e00078b01fffe00000076

The command response in decoded form:

ZigBee Explicit Rx Indicator

API 0x91 64DestAddr 0x0013A2004047B55C 16DestAddr 0xFFFE SrcEP 0xE7 DestEP 0xE6

ClusterID 0x8006 ProfileID 0xC105 Options 0x00

RF_Data 0x09EE010034120441424344

The response in terms of Preamble, ZCL Header, and ZCL payload:

Preamble = "910013a2004047b55cfffee7e68006c10500"

The packet has its endpoint values reversed from the request, and the clusterID is 0x8006 indicating a Group cluster response.

ZCL_header = "09 ee 01"

The first field is a frame control field which specifies a Cluster Specific command (0x01) using a Server->Client direction (0x08). The second field is a transaction sequence number which is used to associate the response with the command request. The third field is the command identifier "View Group" (0x01) [6].

ZCL_payload = "00 3412 0441424344"

The first byte is a status byte (SUCCESS=0x00)[6][4]. The next two bytes hold the groupID (0x1234) in little-endian form. The next byte is the name string length (0x04). The remaining bytes are the ASCII name string characters ("ABCD").

And here is the decoded second message, which is a Tx Status for the original command request. If the FrameId value in the original command request had been zero, or if no space was available in the transmit UART buffer, then no Tx Status message would occur.

ZigBee Tx Status

API 0x8B FrameID 0x01 16DestAddr 0xFFFE

Transmit Retries 0x00 Delivery Status 0x00 Discovery Status 0x00 Success

Get Group Membership (1 of 2)

The purpose of this first form of the Get Group Membership command is to get all the groupIDs associated with a particular endpoint.

The intent of the example is to get all the groupIDs associated with endpoint E7.

The example packet is given in three parts, the preamble, ZCL Header, and ZCL payload:

Preamble = "11 01 "+LocalDevice64Addr+"FFFE E6 E7 0006 C105 00 00"

The packet is addressed to the local node, using a source endpoint of 0xE6, clusterID of 0x0006, and profileID of 0xC105. The destination endpoint E7 holds the endpoint parameter for the "Get Group Membership" command.

ZCL_header = "01 ee 02"

The first field (byte) is a frame control field which specifies a Cluster Specific command (0x02) using a Client->Server direction(0x00). The second field is a transaction sequence number which is used to associate the response with the command request. The third field is the command identifier for "Get Group Membership" (0x02) [7].

ZCL_payload = "00"

The first byte is the group count. If it is zero, then all groupIDs with an endpoint value which matches the given endpoint parameter will be returned in the response.

The example packet in raw hex byte form:

7e001811010013a2004047b55cfffee6e70006c105000001ee020019

The response in raw hex byte form, consisting of two packets:

7e0019910013a2004047b55cfffee7e68006c1050009ee02ff01341235

7e00078b01fffe00000076

The response in decoded form:

ZigBee Explicit Rx Indicator

API 0x91 64DestAddr 0x0013A2004047B55C 16DestAddr 0xFFFE SrcEP 0xE7 DestEP 0xE6

ClusterID 0x8006 ProfileID 0xC105 Options 0x00

RF_Data 0x09EE02FF013412

The response in terms of Preamble, ZCL Header, and ZCL Payload:

Preamble = "910013a2004047b55cfffee7e68006c10500"

The packet has the endpoints reversed from the request, and the clusterID is 0x8006 indicating a Group cluster response.

ZCL_header = "09 ee 02"

The first field is a frame control field which specifies a Cluster Specific command (0x01) using a Server->Client direction (0x08). The second field is a transaction sequence number which is used to associate the response with the command request. The third field is the command identifier "Get Group Membership" (0x02) [8].

ZCL_payload = "FF 01 3412"

The first byte is the remaining capacity of the group table. 0xFF means unknown. The XBee returns this value because the capacity of the group table is dependent on the remaining capacity of the binding table, thus the capacity of the group table is unknown. The second byte is the group count (0x01). The remaining bytes are the groupIDs in little-endian representation.

And here is the decoded second message, which is a Tx Status for the original command request. If the FrameId value in the original command request had been zero, or if no space was available in the transmit UART buffer, then no Tx Status message would occur.

ZigBee Tx Status

API 0x8B FrameID 0x01 16DestAddr 0xFFFE

Transmit Retries 0x00 Delivery Status 0x00 Discovery Status 0x00 Success

Get Group Membership (2 of 2)

The purpose of this second form of the Get Group Membership command is to get the set of groupIDs associated with a particular endpoint which are a subset of a list of given groupIDs.

The intent of the example is to get the groupIDs associated with endpoint E7 which are a subset of a given list of groupIDs (0x1234, 0x5678).

The example packet is given in three parts, the preamble, ZCL Header, and ZCL payload:

Preamble = "11 01 "+LocalDevice64Addr+"FFFE E6 E7 0006 C105 00 00"

The packet is addressed to the local node, using a source endpoint of 0xE6, clusterID of 0x0006, and profileID of 0xC105. The destination endpoint E7 is the endpoint parameter for the "Get Group Membership" command.

ZCL_header = "01 ee 02"

The first field (byte) is a frame control field which specifies a Cluster Specific command (0x02) using a Client->Server direction(0x00). The second field is a transaction sequence number which is used to associate the response with the command request. The third field is the command identifier for "Get Group Membership" (0x02) [7].

ZCL_payload = "02 34127856"

The first byte is the group count. The remaining bytes are a groupIDs which use little-endian representation.

The example packet in raw hex byte form:

7e001c11010013a2004047b55cfffee6e70006c105000001ee02023412785603

The response in raw hex byte form, consisting of two packets:

7e0019910013a2004047b55cfffee7e68006c1050009ee02ff01341235

7e00078b01fffe00000076

The response in decoded form:

ZigBee Explicit Rx Indicator

API 0x91 64DestAddr 0x0013A2004047B55C 16DestAddr 0xFFFE SrcEP 0xE7 DestEP 0xE6

ClusterID 0x8006 ProfileID 0xC105 Options 0x00

RF_Data 0x09EE02FF013412

The response in terms of Preamble, ZCL Header, and ZCL Payload:

Preamble = "910013a2004047b55cfffee7e68006c10500"

The packet has the endpoints reversed from the request, the clusterID is 0x8006 indicating a Group cluster response.

ZCL_header = "09 ee 02"

The first field is a frame control field which specifies a Cluster Specific command (0x01) using a Server->Client direction (0x08). The second field is a transaction sequence number which is used to associate the response with the command request. The third field is the command identifier "Get Group Membership" (0x02) [8].

ZCL_payload = "FF 01 3412"

The first byte is the remaining capacity of the group table. 0xFF means unknown. The XBee returns this value because the capacity of the group table is dependent on the remaining capacity of the binding table, thus the capacity of the group table is unknown. The second byte is the group count (0x01). The remaining bytes are the groupIDs in little-endian representation.

And here is the decoded second message, which is a Tx Status for the original command request. If the FrameId value in the original command request had been zero, or if no space was available in the transmit UART buffer, then no Tx Status message would occur.

ZigBee Tx Status

API 0x8B FrameID 0x01 16DestAddr 0xFFFE

Transmit Retries 0x00 Delivery Status 0x00 Discovery Status 0x00 Success

Remove Group

The purpose of the Remote Group command is to remove a Group Table entry which associates a given endpoint with a given groupID.

The intent of the example is to remove the association of groupID [TBD] with endpoint E7.

The example packet is given in three parts, the preamble, ZCL Header, and ZCL payload:

Preamble = "11 01 "+LocalDevice64Addr+"FFFE E6 E7 0006 C105 00 00"

The packet is addressed to the local node, using a source endpoint of 0xE6, clusterID of 0x0006, and profileID of 0xC105. The destination endpoint E7 is the endpoint parameter for the "Remove Group" command.

ZCL_header = "01 ee 03"

The first field is a frame control field which specifies a Cluster Specific command (0x01) using a Client->Server direction(0x00). The second field is a transaction sequence number which is used to associate the response with the command request. The third field is the command identifier "Remove Group" (0x03) [9].

ZCL_payload = "3412"

The two bytes value is the groupID to be removed in little-endian representation.

The packet in raw hex byte form:

7e001911010013a2004047b55cfffee6e70006c105000001ee033412d2

The response in raw hex byte form, consisting of two packets:

7e0018910013a2004047b55cfffee7e68006c1050009ee0300341235

7e00078b01fffe00000076

The command response in decoded form:

ZigBee Explicit Rx Indicator

API 0x91 64DestAddr 0x0013A2004047B55C 16DestAddr 0xFFFE SrcEP 0xE7 DestEP 0xE6

ClusterID 0x8006 ProfileID 0xC105 Options 0x00

RF_Data 0x09EE03003412

The response in terms of Preamble, ZCL Header, and ZCL payload:

Preamble = "910013a2004047b55cfffee7e68006c10500"

The packet has its endpoint values reversed from the request, and the clusterID is 0x8006 indicating a Group cluster response.

ZCL_header = "09 ee 03"

The first field is a frame control field which specifies a Cluster Specific command (0x01) using a Server->Client direction (0x08). The second field is a transaction sequence number which is used to associate the response with the command request. The third field is the command identifier "Remove Group" (0x03) [10].

ZCL_payload = "00 3412"

The first byte is a status byte (SUCCESS=0x00)[10][4]. The next two bytes is the groupID (0x1234) value in little-endian form.

And here is the decoded second message, which is a Tx Status for the original command request. If the FrameId value in the original command request had been zero, or if no space was available in the transmit UART buffer, then no Tx Status message would occur.

ZigBee Tx Status

API 0x8B FrameID 0x01 16DestAddr 0xFFFE

Transmit Retries 0x00 Delivery Status 0x00 Discovery Status 0x00 Success

Remove All Groups

The purpose of the Remove All Groups command is to clear all entries from the group table which are associated with a target endpoint.

The intent of the example is to remove all groups associated with endpoint E7.

The packet:

Preamble = "11 01 "+LocalDevice64Addr+"FFFE E6 E7 0006 C105 00 00"

The packet is addressed to the local node, using a source endpoint of 0xE6, clusterId of 0x0006, and profileID of 0xC105. The destination endpoint E7 is the endpoint parameter for the "Remove All Groups" command.

ZCL_header = "01 ee 04"

The first field is a frame control field which specifies a Cluster Specific command (0x01) using a Client->Server direction(0x00). The second field is a transaction sequence number which is used to associate the response with the command request. The third field is the command identifier "Remove All Groups" (0x04)[11].

ZCL_payload = ""

No payload is needed for this command.

The packet in raw hex byte form:

7e001711010013a2004047b55cfffee6e70006c105000001ee0417

The response in raw hex byte form, consisting of two packets:

7e0016910013a2004047b55cfffee7e68006c1050009ee04007c

7e00078b01fffe00000076

The command response in decoded form:

ZigBee Explicit Rx Indicator

API 0x91 64DestAddr 0x0013A2004047B55C 16DestAddr 0xFFFE SrcEP 0xE7 DestEP 0xE6

ClusterID 0x8006 ProfileID 0xC105 Options 0x00

RF_Data 0x09ee0400

The response in terms of Preamble, ZCL Header, and ZCL payload.

Preamble = "910013a2004047b55cfffee7e68006c10500"

The packet has its endpoints values reversed from the request, and the clusterID is 0x8006 indicating a Group cluster response.

ZCL_header = "09 ee 04"

The first field is a frame control field which specifies a Cluster Specific command (0x01) using a Server->Client direction (0x08). The second field is a transaction sequence number which is used to associate the response with the command request. The third field is the command identifier "Remove All Groups" (0x04) [10].

ZCL_payload = "00"

The first byte is a status byte (SUCCESS=0x00)[4].

And here is the decoded second message, which is a Tx Status for the original command request. If the FrameID value in the original command request had been zero, or if no space was available in the transmit UART buffer, then no Tx Status message would occur.

ZigBee Tx Status

API 0x8B FrameID 0x01 16DestAddr 0xFFFE

Transmit Retries 0x00 Delivery Status 0x00 Discovery Status 0x00 Success

Default Responses

Many errors are returned as a default response. For example, a RFData payload of a response containing 08010b788b would be decoded as:

ZCL_header = "08 01 03" - general command/server-to-client, transseqnum=1, default_response_command(0x03)

ZCL_payload = "78 8b" - original cmdID, status code (0x8b) EMBER_ZCL_STATUS_NOT_FOUND

Common Status Codes

This section lists some of the more frequently occuring status codes.

0x00 EMBER_ZCL_STATUS_SUCCESS: Command request was successful

0x01 EMBER_ZCL_STATUS_FAILURE: Command request failed - for example, a call to remove an entry from the group table returned an error

0x80 EMBER_ZCL_STATUS_MALFORMED_COMMAND: no RFData in the API frame; ZCL Payload appears truncated from what is expected

0x81 EMBER_ZCL_STATUS_UNSUP_CLUSTER_COMMAND: unexpected direction in the Frame Control Field of the ZCL Header; unexpected command identifier code value in the ZCL header

0x82 EMBER_ZCL_STATUS_UNSUP_GENERAL_COMMAND: unexpected frametype in the Frame Control Field of the ZCL Header

0x84 EMBER_ZCL_STATUS_UNSUP_MANUF_GENERAL_COMMAND: unexpected manufacturer specific indication in the Frame Control Field of the ZCL Header

0x8b EMBER_ZCL_STATUS_NOT_FOUND: An attempt at Get Group Membership or Remove Group could not find a matching entry in the group table

A full set of status codes appears in the documentation [4].

Bibliography

[1] ZigBee Cluster Library, document 075123r02, section 3.6.

The following cross references all appear in the ZCL document:

[2] Add Group Command, section 3.6.2.2.3.

[3] Add Group Response, section 3.6.2.3.1.

[4] Status Enumerations, section 2.5.3.

[5] View Group Command, section 3.6.2.2.4.

[6] View Group Response, section 3.6.2.3.2.

[7] Get Group Membership Command, section 3.6.2.2.5.

[8] Get Group Membership Response, section 3.6.2.3.3.

[9] Remove Group Command, section 3.6.2.2.6.

[10] Remove Group Response, section 3.6.2.3.4.

[11] Remove All Groups Command, section 3.6.2.2.7.

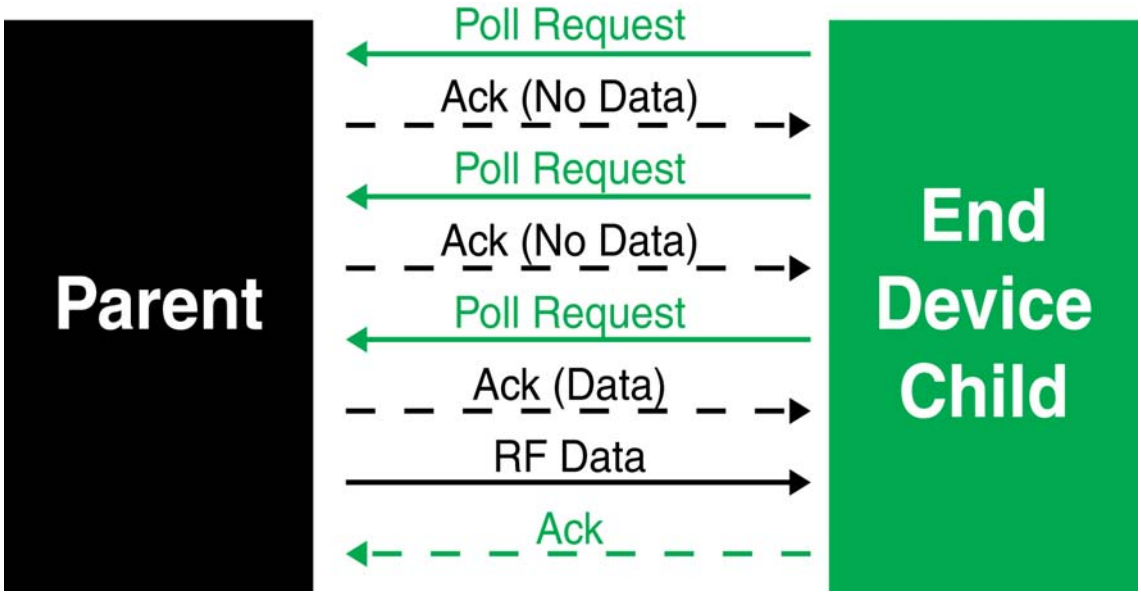
7. Managing End Devices

ZigBee end devices are intended to be battery-powered devices capable of sleeping for extended periods of time. Since end devices may not be awake to receive RF data at a given time, routers and coordinators are equipped with additional capabilities (including packet buffering and extended transmission timeouts) to ensure reliable data delivery to end devices.

End Device Operation

When an end device joins a ZigBee network, it must find a router or coordinator device that is allowing end devices to join. Once the end device joins a network, a parent-child relationship is formed between the end device and the router or coordinator that allowed it to join. See chapter 3 for details.

When the end device is awake, it sends poll request messages to its parent. When the parent receives a poll request, it checks a packet queue to see if it has any buffered messages for the end device. It then sends a MAC layer acknowledgment back to the end device that indicates if it has data to send to the end device or not.



If the end device receives the acknowledgment and finds that the parent has no data for it, the end device can return to idle mode or sleep. Otherwise, it will remain awake to receive the data. This polling mechanism allows the end device to enter idle mode and turn its receiver off when RF data is not expected in order to reduce current consumption and conserve battery life.

The end device can only send data directly to its parent. If an end device must send a broadcast or a unicast transmission to other devices in the network, it sends the message directly to its parent and the parent performs any necessary route or address discoveries to route the packet to the final destination.

Parent Operation

Each router or coordinator maintains a child table that contains the addresses of its end device children. A router or coordinator that has unused entries in its child table is said to have end device capacity, or the ability to allow new end devices to join. If the child table is completely filled (such that the number of its end device children matches the number of child table entries), the device cannot allow any more end devices to join to it.

Since the end device children are not guaranteed to be awake at a given time, the parent is responsible for managing incoming data packets in behalf of its end device children. If a parent receives an RF data transmission destined for one of its end device children, and if the parent has enough unused buffer space, it will buffer the packet. The data packet will remain buffered until a timeout expires, or until the end device sends a poll request to retrieve the data.

The parent can buffer one broadcast transmission for all of its end device children. When a broadcast transmission is received and buffered, the parent sets a flag in its child table when each child polls and retrieves the packet. Once all children have received the broadcast packet, the buffered broadcast packet is discarded. If all children have not received a buffered broadcast packet and a new broadcast is received, the old broadcast packet is discarded, the child table flags are cleared, and the new broadcast packet is buffered for the end device children. This is demonstrated in the figure below.

End Device Child Table

Address	Received Broadcast
Ox2120	T
OxF220	F
OxC100	F
Ox5750	T

When an end device sends data to its parent that is destined for a remote device in the network, the parent buffers the data packet until it can establish a route to the destination. The parent may perform a route or 16-bit address discovery in behalf of its end device children. Once a route is established, the parent sends the data transmission to the remote device.

End Device Poll Timeouts

To better support mobile end devices (end devices that can move around in a network), parent router and coordinator devices have a poll timeout for each end device child. If an end device does not send a poll request to its parent within the poll timeout, the parent will remove the end device from its child table. This allows the child table on a router or coordinator to better accommodate mobile end devices in the network.

Packet Buffer Usage

Packet buffer usage on a router or coordinator varies depending on the application. The following activities can require use of packet buffers for up to several seconds:

- Route and address discoveries
- Application broadcast transmissions
- Stack broadcasts (e.g. ZDO "Device Announce" messages when devices join a network)
- Unicast transmissions (buffered until acknowledgment is received from destination or retries exhausted)
- Unicast messages waiting for end device to wake.

Applications that use regular broadcasting or that require regular address or route discoveries will use up a significant number of buffers, reducing the buffer availability for managing packets for end device children. Applications should reduce the number of required application broadcasts, and consider implementing an external address table or many-to-one and source routing if necessary to improve routing efficiency.

Non-Parent Device Operation

Devices in the ZigBee network treat data transmissions to end devices differently than transmissions to other routers and coordinators. Recall that when a unicast transmission is sent, if a network acknowledgment is not received within a timeout, the device resends the transmission. When transmitting data to remote coordinator or router devices, the transmission timeout is relatively short since these devices are powered and responsive. However, since end devices may sleep for some time, unicast transmissions to end devices use an extended timeout mechanism in order to allow enough time for the end device to wake and receive the data transmission from its parent.

If a non-parent device does not know the destination is an end device, it will use the standard unicast timeout for the transmission. However, provisions exist in the Ember ZigBee stack for the parent to inform the message sender that the destination is an end device. Once the sender discovers the destination device is an end device, future transmissions will use the extended timeout. See the XBee Router / Coordinator Configuration section in this chapter for details.

XBee End Device Configuration

XBee end devices support three different sleep modes:

- Pin Sleep
- Cyclic Sleep
- Cyclic Sleep with pin wake-up

Pin sleep allows an external microcontroller to determine when the XBee should sleep and when it should wake by controlling the Sleep_RQ pin. In contrast, cyclic sleep allows the sleep period and wake times to be configured through the use of AT commands. Cyclic sleep with pin wake-up is the same as cyclic sleep except that the module can be awakened before the sleep period expires by lowering the Sleep_Rq line. The sleep mode is configurable with the SM command.

In both pin and cyclic sleep modes, XBee end devices poll their parent every 100ms while they are awake to retrieve buffered data. When a poll request has been sent, the end device enables the receiver until an acknowledgment is received from the parent. (It generally takes less than 10ms from the time the poll request is sent until the acknowledgment is received.) The acknowledgment indicates if the parent has buffered data for the end device child or not. If the acknowledgment indicates the parent has pending data, the end device will leave the receiver on to receive the data. Otherwise, the end device will turn off the receiver and enter idle mode (until the next poll request is sent) to reduce current consumption (and improve battery life).

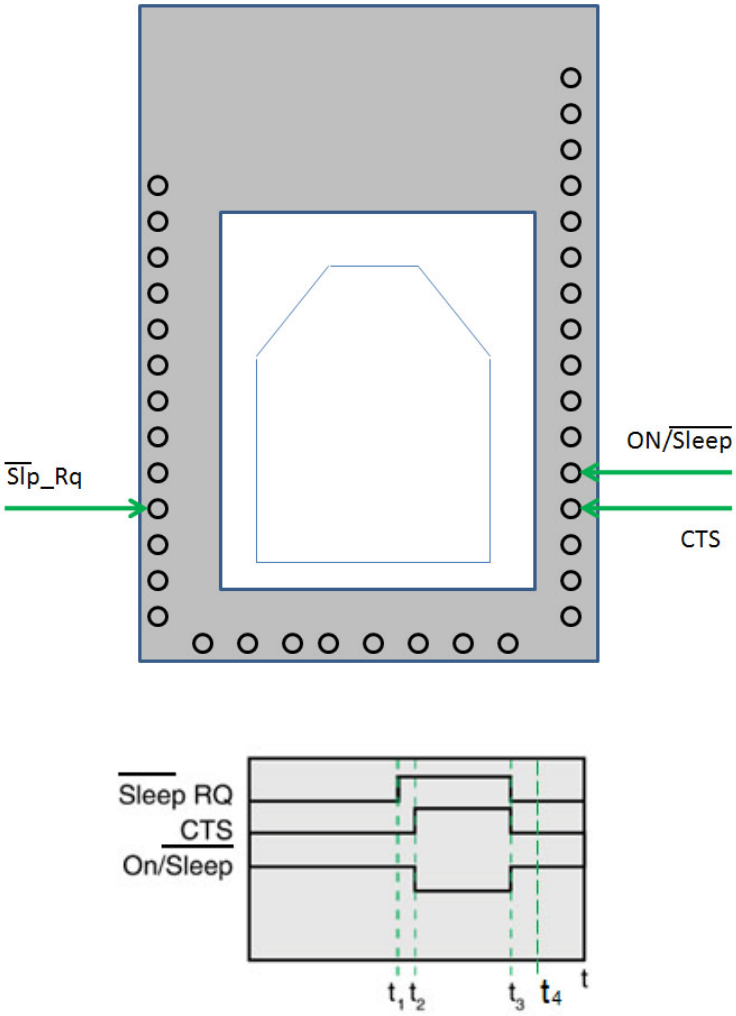
Once the module enters sleep mode, the On/Sleep pin (pin 26) is de-asserted (low) to indicate the module is entering sleep mode. If CTS hardware flow control is enabled (D7 command), the CTS pin (pin 12) is de-asserted (high) when entering sleep to indicate that serial data should not be sent to the module. If the Associate LED pin is configured (D5 command), the associate pin will be driven low to avoid using power to light the LED. Finally, the Sleep_Rq pin will be configured as a pulled-down input so that an external device must drive it high to wake the module. All other pins will be left unmodified during sleep so that they can operate as previously configured by the user. The module will not respond to serial or RF data when it is sleeping. Applications that must communicate serially to sleeping end devices are encouraged to observe CTS flow control.

When the XBee wakes from sleep, the On/Sleep pin is asserted (high), and if flow control is enabled, the CTS pin is also asserted (low). The associate LED and all other pins resume their former configured operation. If the module has not joined a network, it will scan all SC channels after waking to try and find a valid network to join.

Pin Sleep

Pin sleep allows the module to sleep and wake according to the state of the Sleep_RQ pin (pin 9). Pin sleep mode is enabled by setting the SM command to 1.

When Sleep_RQ is asserted (high), the module will finish any transmit or receive operations and enter a low power state. For example, if the module has not joined a network and Sleep_RQ is asserted (high), the module will sleep once the current join attempt completes (i.e. when scanning for a valid network completes). The module will wake from pin sleep when the Sleep_RQ pin is de-asserted (low).



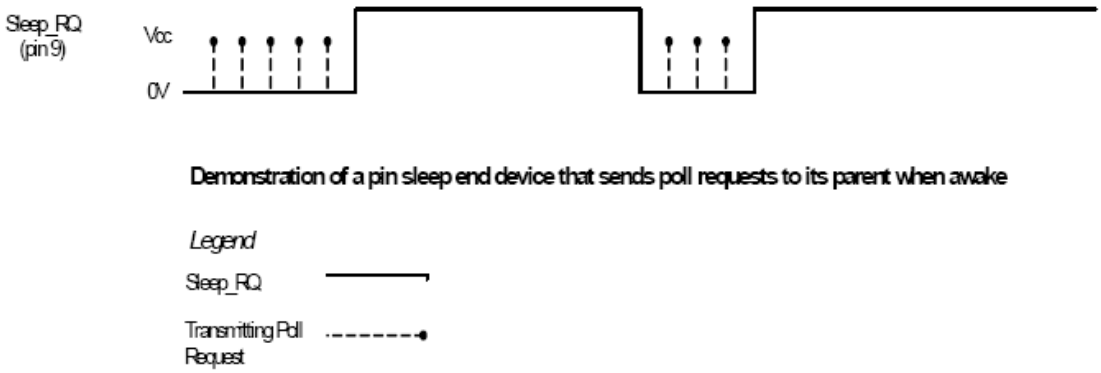
In the figure above, t_1, t_2, t_3 and t_4 represent the following events:

- t_1 - Time when Sleep_RQ is asserted (high)
- t_2 - Time when the XBee enters sleep (CTS state change only if hardware flow control is enabled)
- t_3 - Time when Sleep_RQ is de-asserted (low) and the module wakes.
- t_4 - Time when the module sends a poll request to its parent.

The time between t_1 and t_2 varies depending on the state of the module. In the worst case scenario, if the end device is trying to join a network, or if it is waiting for an acknowledgment from a data transmission, the delay could be up to a few seconds. the time between t3 and t4 is 1-2 ms for a regular module and about 6 ms for a PRO module.

When the XBee is awake and is joined to a network, it sends a poll request to its parent to see if the parent has any buffered data for it. The end device will continue to send poll requests every 100ms while it is awake.

Demonstration of Pin Sleep



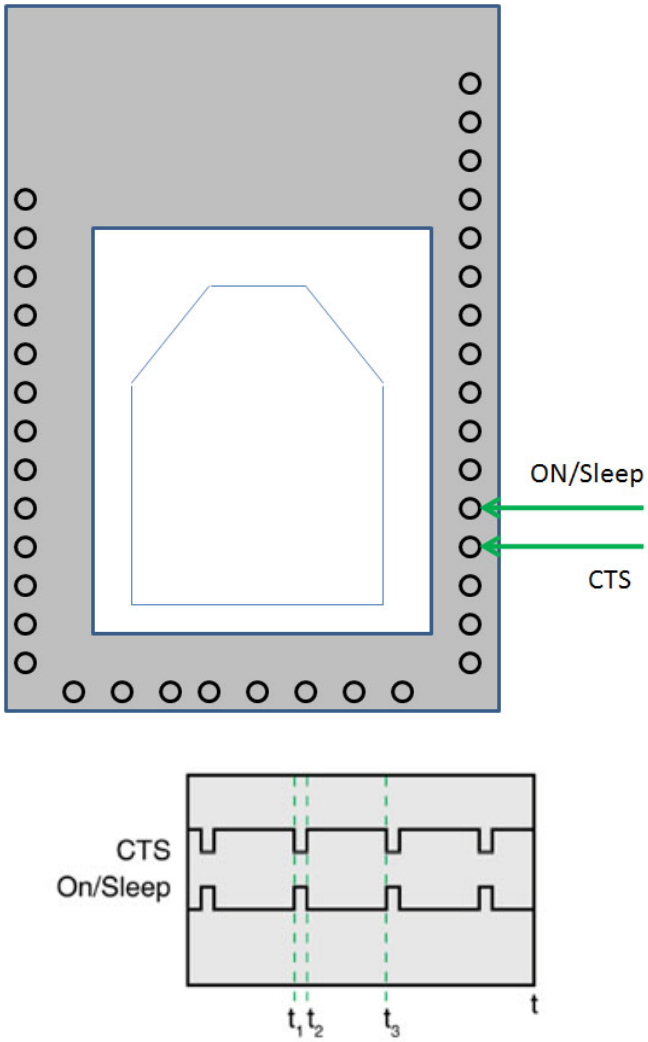
Parent and remote devices must be configured to buffer data correctly and to utilize adequate transmission timeouts. See the XBee Router / Coordinator Configuration section in this chapter for details.

Cyclic Sleep

Cyclic sleep allows the module to sleep for a specified time and wake for a short time to poll its parent for any buffered data messages before returning to sleep again. Cyclic sleep mode is enabled by setting the SM command to 4 or 5. SM5 is a slight variation of SM4 that allows the module to be woken prematurely by asserting the Sleep_RQ pin (pin 10). In SM5, the XBee can wake after the sleep period expires, or if a high-to-low transition occurs on the Sleep_RQ pin. Setting SM to 4 disables the pin wake option.

In cyclic sleep, the module sleeps for a specified time, and then wakes and sends a poll request to its parent to discover if the parent has any pending data for the end device. If the parent has buffered data for the end device, or if serial data is received, the XBee will remain awake for a time. Otherwise, it will enter sleep mode immediately.

The On/Sleep line is asserted (high) when the module wakes, and is de-asserted (low) when the module sleeps. If hardware flow control is enabled (D7 command), the CTS pin will assert (low) when the module wakes and can receive serial data, and de-assert (high) when the module sleeps.



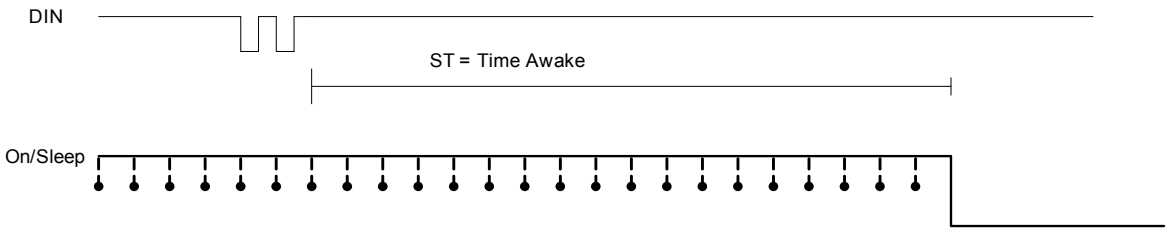
In the figure above, t1, t2, and t3 represent the following events:

- T1 - Time when the module wakes from cyclic sleep
- T2 - Time when the module returns to sleep
- T3 - Later time when the module wakes from cyclic sleep.

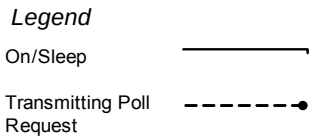
The wake time and sleep time are configurable with software commands as described in the sections below.

Wake Time (Until Sleep)

In cyclic sleep mode (SM=4 or 5), if serial or RF data is received, the module will start a sleep timer (time until sleep). Any data received serially or over the RF link will restart the timer. The sleep timer value is settable with the ST command. While the module is awake, it will send poll request transmissions every 100ms to check its parent for buffered data messages. The module returns to sleep when the sleep timer expires, or if the SI command is sent to it. The following image shows this behavior.



A cyclic sleep end device enters sleep mode when no serial or RF data is received for ST time .



Sleep Period

The sleep period is configured based on the SP, SN, and SO commands. The following table lists the behavior of these commands.

Command	Range	Description
SP	0x20 - 0xAF0 (x 10 ms) (320 - 28,000 ms)	Configures the sleep period of the module.
SN	1 - 0xFFFF	Configures the number of sleep periods multiplier.
SO	0 - 0xFF	Defines options for sleep mode behavior. 0x02 - Always wake for full ST time 0x04 - Enable extended sleep (sleep for full (SP * SN) time)

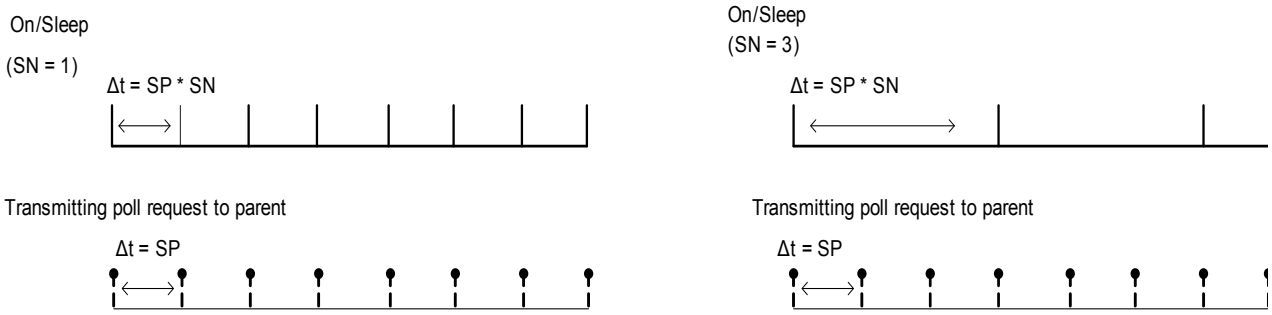
The XBee module supports both a short cyclic sleep and an extended cyclic sleep that make use of these commands. These two modes allow the sleep period to be configured according to the application requirements.

Short Cyclic Sleep

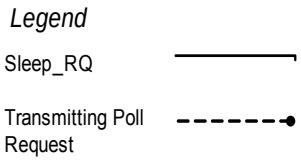
In short cyclic sleep mode, the sleep behavior of the module is defined by the SP and SN commands, and the SO command must be set to 0x00 (default) or 0x02. In short cyclic sleep mode, the SP command defines the sleep period and is settable up to 28 seconds. When the XBee enters short cyclic sleep, it remains in a low power state until the SP time has expired.

After the sleep period expires, the XBee sends a poll request transmission to its parent to determine if its parent has any buffered data waiting for the end device. Since router and coordinator devices can buffer data for end device children up to 30 seconds, the SP range (up to 28 seconds) allows the end device to poll regularly enough to receive buffered data. If the parent has data for the end device, the end device will start its sleep timer (ST) and continue polling every 100ms to receive data. If the end device wakes and finds that its parent has no data for it, the end device can return to sleep immediately.

The SN command can be used to control when the On/Sleep line is asserted (high). If SN is set to 1 (default), the On/Sleep line will be set high each time the XBee wakes from sleep. Otherwise, if SN is greater than 1, the On/Sleep line will only be set high if RF data is received, or after SN wake cycles occur. This allows an external device to remain powered off until RF data is received, or until a number of sleep periods have expired (SN sleep periods). This mechanism allows the XBee to wake at regular intervals to poll its parent for data without waking an external device for an extended time (SP * SN time). This is shown in the figure below.



Setting SN > 1 allows the XBee to silently poll for data without asserting On /Sleep. If RF data is received when polling, On/Sleep will immediately assert .



Note: SP controls the packet buffer time on routers and coordinators. SP should be set on all router and coordinator devices to match the longest end device SP time. See the XBee Router / Coordinator Configuration section for details.

Extended Cyclic Sleep

In extended cyclic sleep operation, an end device can sleep for a multiple of SP time which can extend the sleep time up to several days. The sleep period is configured using the SP and SN commands. The total sleep period is equal to (SP * SN) where SP is measured in 10ms units. The SO command must be set correctly to enable extended sleep.

Since routers and coordinators can only buffer incoming RF data for their end device children for up to 30 seconds, if an end device sleeps longer than 30 seconds, devices in the network need some indication when an end device is awake before they can send data to it. End devices that use extended cyclic sleep should send a transmission (such as an IO sample) when they wake to inform other devices that they are awake and can receive data. It is recommended that extended sleep end devices set SO to wake for the full ST time in order to provide other devices with enough time to send messages to the end device.

Similar to short cyclic sleep, end devices running in this mode will return to sleep when the sleep timer expires, or when the SI command is received.

Transmitting RF Data

An end device may transmit data when it wakes from sleep and has joined a network. End devices transmit directly to their parent and then wait for an acknowledgment to be received. The parent will perform any required address and route discoveries to help ensure the packet reaches the intended destination before reporting the transmission status to the end device.

Receiving RF Data

After waking from sleep, an end device sends a poll request to its parent to determine if the parent has any buffered data for it. In pin sleep mode, the end device polls every 100ms while the Sleep_RQ pin is de-asserted (low). In cyclic sleep mode, the end device will only poll once before returning to sleep unless the sleep timer (ST) is started (serial or RF data is received). If the sleep timer is started, the end device will continue to poll every 100ms until the sleep timer expires.

This firmware includes an adaptive polling enhancement where, if an end device receives RF data from its parent, it sends another poll after a very short delay to check for more data. The end device continues to poll at a faster rate as long as it receives data from its parent. This feature greatly improves data throughput to end devices. When the end device no longer receives data from its parent, it resumes polling every 100ms.

I/O Sampling

End devices can be configured to send one or more I/O samples when they wake from sleep. To enable I/O sampling on an end device, the IR command must be set to a non-zero value, and at least one analog or digital I/O pin must be enabled for sampling (D0 - D9, P0-P4 commands). If I/O sampling is enabled, an end device sends an I/O sample when it wakes and starts the ST timer. It will continue sampling at the IR rate until the sleep timer (ST) has expired. See chapter 8 for details.

Waking End Devices with the Commissioning Pushbutton

If the commissioning pushbutton functionality is enabled (D0 command), a high-to-low transition on the AD0/DIO0 pin (pin 33) will cause an end device to wake for 30 seconds. See the Commissioning Pushbutton section in chapter 7 for details.

Parent Verification

Since an end device relies on its parent to maintain connectivity with other devices in the network, XBee end devices include provisions to verify its connection with its parent. End devices monitor their link with their parent when sending poll messages and after a power cycle or reset event as described below.

When an end device wakes from sleep, it sends a poll request to its parent. In cyclic sleep, if RF or serial data is not received and the sleep timer is not started, the end device polls one time and returns to sleep for another sleep period. Otherwise, the end device continues polling every 100ms. If the parent does not send an acknowledgment response to three consecutive poll request transmissions, the end device assumes the parent is out of range, and attempts to find a new parent.

After a power-up or reset event, the end device does an orphan scan to locate its parent. If the parent does not send a response to the orphan scan, the end device attempts to find a new parent.

Rejoining

Once all devices have joined a ZigBee network, the permit-joining attribute should be disabled such that new devices are no longer allowed to join the network. Permit-joining can be enabled later as needed for short times. This provides some protection in preventing other devices from joining a live network.

If an end device cannot communicate with its parent, the end device must be able to join a new parent to maintain network connectivity. However, if permit-joining is disabled in the network, the end device will not find a device that is allowing new joins.

To overcome this problem, ZigBee supports rejoining, where an end device can obtain a new parent in the same network even if joining is not enabled. When an end device joins using rejoining, it performs a PAN ID scan to discover nearby networks. If a network is discovered that has the same 64-bit PAN ID as the end device, it will join the network by sending a rejoin request to one of the discovered devices. The device that receives the rejoin request will send a rejoin response if it can allow the device to join the network (i.e. child table not full). The rejoin mechanism can be used to allow a device to join the same network even if permit-joining is disabled.

To enable rejoining, NJ should be set less than 0xFF on the device that will join. If NJ < 0xFF, the device assumes the network is not allowing joining and first tries to join a network using rejoining. If multiple rejoining attempts fail, or if NJ=0xFF, the device will attempt to join using association.

XBee Router/Coordinator Configuration

XBee routers and coordinators may require some configuration to ensure the following are set correctly:

- RF packet buffering timeout
- Child poll timeout
- Transmission timeout.

The value of these timeouts depends on the sleep time used by the end devices. Each of these timeouts are discussed below.

RF Packet Buffering Timeout

When a router or coordinator receives an RF data packet intended for one of its end device children, it buffers the packet until the end device wakes and polls for the data, or until a packet buffering timeout occurs. This timeout is settable using the SP command. The actual timeout is (1.2 * SP), with a minimum timeout of 1.2 seconds and a maximum of 30 seconds. Since the packet buffering timeout is set slightly larger than the SP setting, SP should be set the same on routers and coordinators as it is on cyclic sleep end devices. For pin sleep devices, SP should be set as long as the pin sleep device can sleep, up to 30 seconds.

Note: In pin sleep and extended cyclic sleep, end devices can sleep longer than 30 seconds. If end devices sleep longer than 30 seconds, parent and non-parent devices must know when the end device is awake in order to reliably send data. For applications that require sleeping longer than 30 seconds, end devices should transmit an IO sample or other data when they wake to alert other devices that they can send data to the end device.

Child Poll Timeout

Router and coordinator devices maintain a timestamp for each end device child indicating when the end device sent its last poll request to check for buffered data packets. If an end device does not send a poll request to its parent for a certain period of time, the parent will assume the end device has moved out of range and will remove the end device from its child table. This allows routers and coordinators to be responsive to changing network conditions. The NC command can be issued at any time to read the number of remaining (unused) child table entries on a router or coordinator.

The child poll timeout is settable with the SP and SN commands. SP and SN should be set such that SP * SN matches the longest expected sleep time of any end devices in the network. The actual timeout is calculated as (3 * SP * SN), with a minimum of 5 seconds. For networks consisting of pin sleep end devices, the SP and SN values on the coordinator and routers should be set such that SP * SN matches the longest expected sleep period of any pin sleep device. The 3 multiplier ensures the end device will not be removed unless 3 sleep cycles pass without receiving a poll request. The poll timeout is settable up to a couple of months.

Adaptive Polling

The PO command determines the regular polling rate. However, if RF data has been recently received by an end device, it is likely that yet more RF data could be on the way. Therefore, the end device will poll at a faster rate, gradually decreasing its adaptive poll rate until polling resumes at the regular rate as defined by the PO command.

Transmission Timeout

As mentioned in chapter 4, when sending RF data to a remote router, since routers are always on, the timeout is based on the number of hops the transmission may traverse. This timeout it settable using the NH command. (See chapter 4 for details.)

Since end devices may sleep for lengthy periods of time, the transmission timeout to end devices also includes some allowance for the sleep period of the end device. When sending data to a remote end device, the transmission timeout is calculated using the SP and NH commands. If the timeout occurs and an acknowledgment has not been received, the source device will resend the transmission until an acknowledgment is received, up to two more times.

The transmission timeout per attempt is:

3 * ((unicast router timeout) + (end device sleep time)), or

3 * ((50 * NH) + (1.2 * SP)), where SP is measured in 10ms units.

Putting It All Together

Short Sleep Periods

Pin and cyclic sleep devices that sleep less than 30 seconds can receive data transmissions at any time since their parent device(s) will be able to buffer data long enough for the end devices to wake and poll to receive the data. SP should be set the same on all devices in the network. If end devices in a network have more than one SP setting, SP on the routers and coordinators should be set to match the largest SP setting of any end device. This will ensure the RF packet buffering, poll timeout, and transmission timeouts are set correctly.

Extended Sleep Periods

Pin and cyclic sleep devices that might sleep longer than 30 seconds cannot receive data transmissions reliably unless certain design approaches are taken. Specifically, the end devices should use IO sampling or another mechanism to transmit data when they wake to inform the network they can receive data. SP and SN should be set on routers and coordinators such that (SP * SN) matches the longest expected sleep time. This configures the poll timeout so end devices are not expired from the child table unless a poll request is not received for 3 consecutive sleep periods.

As a general rule of thumb, SP and SN should be set the same on all devices in almost all cases.

Sleep Examples

This section covers some sample XBee configurations to support different sleep modes. Several AT commands are listed with suggested parameter values. The notation in this section includes an '=' sign to indicate what each command register should be set to - for example, SM=4. This is not the correct notation for setting command values in the XBee. In AT command mode, each command is issued with a leading 'AT' and no '=' sign - for example ATSM4. In the API, the two byte command is used in the command field, and parameters are populated as binary values in the parameter field.

Example 1: Configure a device to sleep for 20 seconds, but set SN such that the On/Sleep line will remain de-asserted for up to 1 minute.

The following settings should be configured on the end device.

SM = 4 (cyclic sleep) or 5 (cyclic sleep, pin wake)

SP = 0x7D0 (2000 decimal). This causes the end device to sleep for 20 seconds since SP is measured in units of 10ms.

SN = 3. (With this setting, the On/Sleep pin will assert once every 3 sleep cycles, or when RF data is received)

SO = 0

All router and coordinator devices on the network should set SP to match SP on the end device. This ensures that RF packet buffering times and transmission timeouts will be set correctly.

Since the end device wakes after each sleep period (ATSP), the SN command can be set to 1 on all routers and the coordinator.

Example 2: Configure an end device to sleep for 20 seconds, send 4 IO samples in 2 seconds, and return to sleep.

Since SP is measured in 10ms units, and ST and IR are measured in 1ms units, configure an end device with the following settings:

SM = 4 (cyclic sleep) or 5 (cyclic sleep, pin wake)

SP = 0x7D0 (2000 decimal). This causes the end device to sleep for 20 seconds.

SN = 1

SO = 0

ST = 0x7D0 (2000 decimal). This sets the sleep timer to 2 seconds.

IR = 0x258 (600 decimal). Set IR to a value greater than (2 seconds / 4) to get 4 samples in 2 seconds. The end device sends an IO sample at the IR rate until the sleep timer has expired.

At least one analog or digital IO line must be enabled for IO sampling to work. To enable pin 32 (AD1/DIO1) as a digital input line, the following must be set:

D1 = 3

All router and coordinator devices on the network should set SP to match SP on the end device. This ensures that RF packet buffering times and transmission timeouts will be set correctly.

Example 3: Configure a device for extended sleep: to sleep for 4 minutes.

SP and SN must be set such that SP * SN = 4 minutes. Since SP is measured in 10ms units, the following settings can be used to obtain 4 minute sleep.

SM = 4 (cyclic sleep) or 5 (cyclic sleep, pin wake)

SP = 0x7D0 (2000 decimal, or 20 seconds)

SN = 0x0B (12 decimal)

SO = 0x04 (enable extended sleep)

With these settings, the module will sleep for SP * SN time, or (20 seconds * 12) = 240 seconds = 4 minutes.

For best results, the end device should send a transmission when it wakes to inform the coordinator (or network) when it wakes. It should also remain awake for a short time to allow devices to send data to it. The following are recommended settings.

ST = 0x7D0 (2 second wake time)

SO = 0x06 (enable extended sleep and wake for ST time)

IR = 0x800 (send 1 IO sample after waking). At least one analog or digital IO sample should be enabled for IO sampling.

With these settings, the end device will wake after 4 minutes and send 1 IO sample. It will then remain awake for 2 seconds before returning to sleep.

SP and SN should be set to the same values on all routers and coordinators that could allow the end device to join. This will ensure the parent does not timeout the end device from its child table too quickly.

The SI command can optionally be sent to the end device to cause it to sleep before the sleep timer expires.

8. XBee Analog and Digital I/O Lines

XBee ZB firmware supports a number of analog and digital I/O pins that are configured through software commands. Analog and digital I/O lines can be set or queried. The following table lists the configurable I/O pins and the corresponding configuration commands.

Module Pin Names	Module Pin	AT Command	Command Range
DOUT/DIO13	3	P3	0, 1, 3-5
DIN/$\overline{\text{CONFIG}}$/DIO14	4	P4	0, 1, 3-5
PWM RSSI/DIO10	7	P0	0, 1, 3-5
PWM1/DIO11	8	P1	0, 1, 3-5
DTR/Slp_Rq/DIO8	10	D8	0, 1, 3-5
PTI_DATA/SPI_$\overline{\text{Attn}}$/ADC5/ DIO19	12	P9	0, 1, 6
SPI_SClk/DIO18	14	P8	0, 1
SPI_$\overline{\text{SSel}}$/DIO17	15	P7	0, 1
SPI_MOSI/DIO16	16	P6	0, 1
SPI_MISO/DIO15	17	P5	0,1
JTMS/SWDIO/DIO12/CD	21	P2	0, 3-5
JTRst/DIO4	24	D4	0, 3-5
$\overline{\text{CTS}}$/DIO7	25	D7	0, 1, 3-7
JTDO/On_$\overline{\text{SLP}}$/DIO9	26	D9	0, 1, 3-5
JTDI/Assoc/DIO5	28	D5	0, 1, 3-5
RTS/DIO6/SClk2	29	D6	0, 1, 3-5
AD3/DIO3	30	D3	0, 2-5
AD2/DIO2	31	D2	0, 2-5
PTI_En/AD1/DIO1	32	D1	0, 2-6
AD0/DIO0/Comm	33	D0	0-5

I/O Configuration

To enable an analog or digital I/O function on one or more XBee module pin(s), the appropriate configuration command must be issued with the correct parameter. After issuing the configuration command, changes must be applied on the module for the I/O settings to take effect.

Pin Command Parameter	Description
0	Disabled. (See below)
1	Peripheral control
2	Analog
3	Data in monitored. (See below)

Pin Command Parameter	Description
4	Data out default low
5	Data out default high
6	RS485 enable low / packet trace interface
7	RS485 enable high
>7	Unsupported

When the pin command parameter is a 0 or a 3, it operates the same on this platform, except that the pin will not be monitored by I/O sampling if the parameter is 0.

Inputs have three variations:

- floating
- pulled-up
- pulled-down

A floating input is appropriate if the pin is attached to an output that always drives the line. In this case, a pull-up or pull-down resistor would cause more current to be drawn.

A pulled-up input is useful where there might not always be an external source to drive the pin and it is desirable to have the line read high in the absence of an external driver.

Likewise, a pulled-down input is useful when there is not always an external source to drive the pin and it is desirable to have the line read low in the absence of an external driver.

Two commands are available to configure the input type:

- PR determines whether or not an input is pulled. If the corresponding bit in PR is set, then the signal will be pulled. If it is clear, then the signal will be floating.
- PD determines the pull direction. It only applies when the corresponding bit in PR is set. The bit in PD should be set to enable an internal pull-up resistor. It should be cleared to enable an internal pull-down resistor.

I/O Sampling

The XBee ZB modules have the ability to monitor and sample the analog and digital I/O lines. I/O samples can be read locally or transmitted to a remote device to provide indication of the current I/O line states. API mode must be enabled on the receiving device in order to send I/O samples out the serial port. If this mode is not enabled, then remote I/O samples will be discarded

There are three ways to obtain I/O samples, either locally or remotely:

- Queried Sampling
- Periodic Sampling
- Change Detection Sampling.

IO sample data is formatted as shown in the table below

Bytes	Name	Description
1	Sample Sets	Number of sample sets in the packet. (Always set to 1.)

Bytes	Name	Description
2	Digital Channel Mask	Indicates which digital IO lines have sampling enabled. Each bit corresponds to one digital IO line on the module. • bit 0 = AD0/DIO0 • bit 1 = AD1/DIO1 • bit 2 = AD2/DIO2 • bit 3 = AD3/DIO3 • bit 4 = DIO4 • bit 5 = ASSOC/DIO5 • bit 6 = RTS/DIO6 • bit 7 = CTS/GPIO7 • bit 8 = Slp_Rq/DIO8 • bit 9 = On_$\overline{Slp}$/DIO9 • bit 10 = RSSI/DIO10 • bit 11 = PWM/DIO11 • bit 12 = CD/DIO12 • bit 13 = DOUT/DIO13 • bit 14 = DIN/DIO14 For example, a digital channel mask of 0x002F means DIO0,1,2,3, and 5 are enabled as digital I/O.
1	Analog Channel Mask	Indicates which lines have analog inputs enabled for sampling. Each bit in the analog channel mask corresponds to one analog input channel. • bit 0 = AD0/DIO0 • bit 1 = AD1/DIO1 • bit 2 = AD2/DIO2 • bit 3 = AD3/DIO3 • bit 7 = Supply Voltage
Variable	Sampled Data Set	A sample set consisting of 1 sample for each enabled ADC and/or DIO channel, which has voltage inputs of 1143.75 and 342.1875mV. If any digital I/O lines are enabled, the first two bytes of the data set indicate the state of all enabled digital I/O. Only digital channels that are enabled in the Digital Channel Mask bytes have any meaning in the sample set. If no digital I/O are enabled on the device, these 2 bytes will be omitted. Following the digital I/O data (if any), each enabled analog channel will return 2 bytes. The data starts with AIN0 and continues sequentially for each enabled analog input channel up to AIN3, and the supply voltage (if enabled) at the end.

The sampled data set will include 2 bytes of digital I/O data only if one or more I/O lines on the device are configured as digital I/O. If no pins are configured as digital IO, these 2 bytes will be omitted. Pins are configured as digital I/O by setting them to a value of 3, 4, or 5.

The digital I/O data is only relevant if the same bit is enabled in the digital I/O mask.

Analog samples are returned as 10-bit values. The analog reading is scaled such that 0x0000 represents 0 V, and 0x3FF = 1.2 V. (The analog inputs on the module cannot read more than 1.2 V.) Analog samples are returned in order starting with AIN0 and finishing with AIN3, and the supply voltage. Only enabled analog input channels return data as shown in the figure below.

To convert the A/D reading to mV, do the following:

AD(mV) = (A/D reading * 1200mV) / 1024

The reading in the sample frame represents voltage inputs of 1143.75 and 342.1875 mV for AD0 and AD1 respectively.

Queried Sampling

The IS command can be sent to a device locally, or to a remote device using the API remote command frame (see chapter 8 for details). When the IS command is sent, the receiving device samples all enabled digital IO and analog input channels and returns an IO sample. If IS is sent locally, the IO sample is sent out the serial port. If the IS command was received as a remote command, the IO sample is sent over-the-air to the device that sent the IS command.

If the IS command is issued in command mode, the module returns a carriage return-delimited list containing the above-listed fields. If the IS command is issued in API mode, an API command response contains the same information.

The following table shows an example of the fields in an IS response.

Example	Sample AT Response
0x01	[1 sample set]
0x0C0C	[Digital Inputs: DIO 2, 3, 10, 11 low]
0x03	[Analog Inputs: A/D 0, 1]
0x0408	[Digital input states: DIO 3, 10 high, DIO 2, 11 low]
0x03D0	[Analog input ADIO 0= 0x3D0]
0x0124	[Analog input ADIO 1=0x120]

Periodic I/O Sampling

Periodic sampling allows an XBee module to take an I/O sample and transmit it to a remote device at a periodic rate. The periodic sample rate is set by the IR command. If IR is set to 0, periodic sampling is disabled. For all other values of IR, data will be sampled after IR milliseconds have elapsed and transmitted to a remote device. The DH and DL commands determine the destination address of the I/O samples. DH and DL can be set to 0 to transmit to the coordinator, or to the 64-bit address of the remote device (SH and SL). Only devices running in API mode can send I/O data samples out their serial port. Devices running in transparent mode will discard received I/O data samples.

A sleeping end device will transmit periodic IO samples at the IR rate until the ST timer expires and the device can resume sleeping.

Change Detection Sampling

Modules can be configured to transmit a data sample immediately whenever a monitored digital I/O pin changes state. The IC command is a bitmask that can be used to set which digital I/O lines should be monitored for a state change. If one or more bits in IC is set, an I/O sample will be transmitted as soon as a state change is observed in one of the monitored digital IO lines. Change detection samples are transmitted to the 64-bit address specified by DH and DL.

RSSI PWM

The XBee module features an RSSI/PWM pin (pin 7) that, if enabled, will adjust the PWM output to indicate the signal strength of the last received packet. The P0 (P-zero) command is used to enable the RSSI pulse width modulation (PWM) output on the pin. If P0 is set to 1 (and P1 is not set to 1), the RSSI/PWM pin will output a pulse width modulated signal where the frequency is adjusted based on the received signal strength of the last packet. Otherwise, for all other P0 settings, the pin can be used for general purpose IO.

When a data packet is received, if P0 is set to enable the RSSI/PWM feature, the RSSI PWM output is adjusted based on the RSSI of the last packet. The RSSI/PWM output will be enabled for a time based on the RP command. Each time an RF packet is received, the RSSI/PWM output is adjusted based on the RSSI of the new packet, and the RSSI timer is reset. If the RSSI timer expires, the RSSI/PWM pin is driven low. RP is measured in 100ms units and defaults to a value of 40 (4 seconds).

The RSSI PWM runs at 12MHz and has 2400 total counts (200us period).

RSSI (in dBm) is converted to PWM counts using the following equation:

PWM counts = (41 * RSSI_Unsigned) - 5928

I/O Examples

Example 1: Configure the following I/O settings on the XBee.

Configure AD1/DIO1 as a digital input with pullup resistor enabled

Configure AD2/DIO2 as an analog input

Configure DIO4 as a digital output, driving high.

To configure AD1/DIO1 as an input, issue the ATD1 command with a parameter of 3 ("ATD13"). To enable pull-up resistors on the same pin, the PR command should be issued with bit 3 set (e.g. ATPR8, ATPR1FFF, etc.).

The ATD2 command should be issued with a parameter of 2 to enable the analog input ("ATD22"). Finally, DIO4 can be set as an output, driving high by issuing the ATD4 command with a parameter value of 5 ("ATD45").

After issuing these commands, changes must be applied before the module IO pins will be updated to the new states. The AC or CN commands can be issued to apply changes (e.g. ATAC).

Example 2: Calculate the PWM counts for a packet received with an RSSI of -84dBm.

RSSI = -84 = 0xAC = 172 decimal (unsigned)

PWM counts = (41 * 172) - 5928

PWM counts = 1124

With a total of 2400 counts, this yields an ON time of (1124 / 2400) = 46.8%

Example 3: Configure the RSSI/PWM pin to operate for 2 seconds after each received RF packet.

First, ensure the RSSI/PWM functionality is enabled by reading the P0 (P-zero) command. It should be set to 1 (default).

To configure the duration of the RSSI/PWM output, set the RP command. To achieve a 2 second PWM output, set RP to 0x14 (20 decimal, or 2 seconds) and apply changes (AC command).

After applying changes, all received RF data packets should set the RSSI timer for 2 seconds.

PWM1

When P1 is configured for peripheral operation by setting the value to 1, it outputs a 50% duty cycle PWM with a clock rate of 32,787 Hz, which is a period of 30.5 μs. The main purpose of the PWM output is to provide a clock for the PLUS processor, although it may also be used for other purposes.

*When this feature is enabled, the RSSI PWM output is automatically disabled, even if it is configured.

9. API Operation

As an alternative to Transparent Operation, API (Application Programming Interface) Operations are available. 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 serial port Data Frame.

Please note that Digi may add new API frames to future versions of firmware, so please build into your software interface the ability to filter out additional API frames with unknown Frame Types.

API Frame Specifications

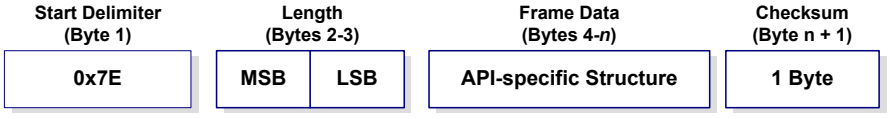
Two API modes are supported and both can be enabled using the AP (API Enable) command. Use the following AP parameter values to configure the module to operate in a particular mode:

- AP = 1: API Operation
- AP = 2: API Operation (with escaped characters)

API Operation (AP parameter = 1)

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

Serial Port Data Frame Structure:



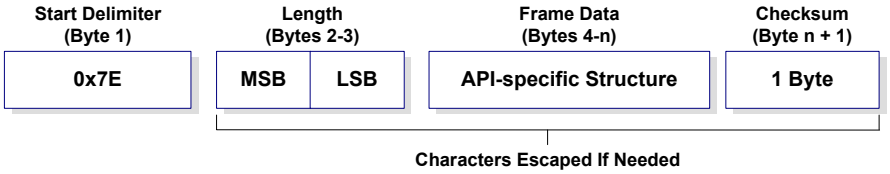
MSB = Most Significant Byte, LSB = Least Significant Byte

Any data received prior to the start delimiter is silently discarded. If the frame is not received correctly or if the checksum fails, the module will reply with a module status frame indicating the nature of the failure.

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

This mode is only available on the UART, not on the SPI serial port. When this API mode is enabled (AP = 2), the UART data frame structure is defined as follows:

UART Data Frame Structure - with escape control characters:



MSB = Most Significant Byte, LSB = Least Significant Byte

Escape characters. When sending or receiving a UART data frame, specific data values must be escaped (flagged) so they do not interfere with the data frame sequencing. To escape an interfering data byte, insert 0x7D and follow it with the byte to be escaped XOR'd with 0x20.

Data bytes that need to be escaped:

- 0x7E – Frame Delimiter
- 0x7D – Escape
- 0x11 – XON
- 0x13 – XOFF

Example - Raw UART Data Frame (before escaping interfering bytes):
0x7E 0x00 0x02 0x23 0x11 0xCB

0x11 needs to be escaped which results in the following frame:
0x7E 0x00 0x02 0x23 0x7D 0x31 0xCB

Note: In the above example, the length of the raw data (excluding the checksum) is 0x0002 and the checksum of the non-escaped data (excluding frame delimiter and length) is calculated as:
0xFF - (0x23 + 0x11) = (0xFF - 0x34) = 0xCB.

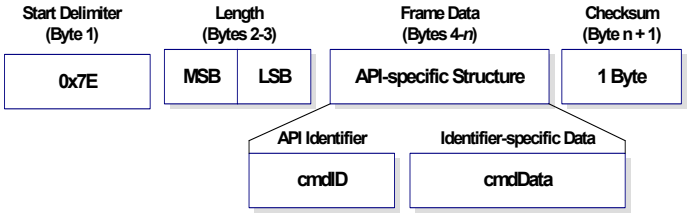
Length

The length field has a two-byte value that specifies the number of bytes that will be contained in the frame data field. It does not include the checksum field.

Frame Data

Frame data of the serial port data frame forms an API-specific structure as follows:

Serial Port Data Frame & API-specific Structure:



The cmdID frame (API-identifier) indicates which API messages will be contained in the cmdData frame (Identifier-specific data). Note that multi-byte values are sent big endian.The XBee modules support the following API frames:

API Frame Names and Values

API Frame Names	API ID
AT Command	0x08
AT Command - Queue Parameter Value	0x09
ZigBee Transmit Request	0x10
Explicit Addressing ZigBee Command Frame	0x11
Remote Command Request	0x17
Create Source Route	0x21
AT Command Response	0x88
Modem Status	0x8A
ZigBee Transmit Status	0x8B
ZigBee Receive Packet (AO=0)	0x90
ZigBee Explicit Rx Indicator (AO=1)	0x91
ZigBee IO Data Sample Rx Indicator	0x92
XBee Sensor Read Indicator (AO=0)	0x94
Node Identification Indicator (AO=0)	0x95
Remote Command Response	0x97
Over-the-Air Firmware Update Status	0xA0
Route Record Indicator	0xA1
Many-to-One Route Request Indicator	0xA3

Checksum

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

To calculate: Not including frame delimiters and length, add all bytes keeping only the lowest 8 bits of the result and subtract the result from 0xFF.

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

API Examples

Example: Create an API AT command frame to configure an XBee to allow joining (set NJ to 0xFF).

The frame should look like:

0x7E 0x00 0x05 0x08 0x01 0x4E 0x4A 0xFF 5F

Where 0x0005 = length

0x08 = AT Command API frame type

0x01 = Frame ID (set to non-zero value)

0x4E4A = AT Command ('NJ')

0xFF = value to set command to

0x5F = Checksum

The checksum is calculated as [0xFF - (0x08 + 0x01 + 0x4E + 0x4A + 0xFF)]

Example: Send an ND command to discover the devices in the PAN.

The frame should look like:

0x7E 0x00 0x04 0x08 0x01 0x4E 0x44 0x64

Where 0x0004 = length

0x08 = AT Command API frame type

0x01 = Frame ID (set to non-zero value)

0x4E44 = AT command ('ND')

0x64 = Checksum

The checksum is calculated as [0xFF - (0x08 + 0x01 + 0x4E + 0x44)]

Example: Send a remote command to the coordinator to set AD1/DIO1 as a digital input (D1=3) and apply changes to force the IO update.

The API remote command frame should look like:

0x7E 0x00 0x10 0x17 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0xFF 0xFE 0x02 0x44 0x31 0x03 0x70

Where

0x10 = length (16 bytes excluding checksum)

0x17 = Remote Command API frame type

0x01 = Frame ID

0x0000000000000000 = Coordinator's address (can be replaced with coordinator's actual 64-bit address if known)

0xFFFE = 16- bit Destination Address

0x02 = Apply Changes (Remote Command Options)

0x4431 = AT command ('D1')

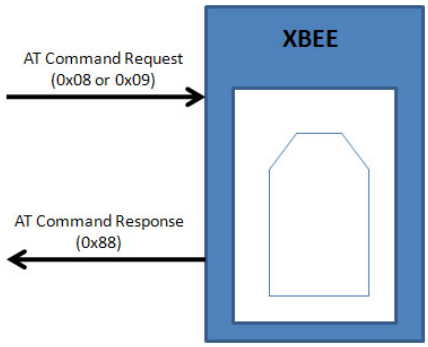
0x03 = Command Parameter (the parameter could also be sent as 0x0003 or 0x00000003)

0x70 = Checksum

API Serial Port Exchanges

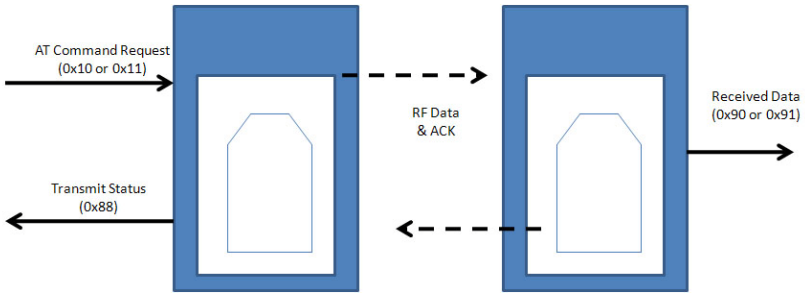
AT Commands

The following image shows the API frame exchange that takes place at the serial port when sending an AT command request to read or set a module parameter. The response can be disabled by setting the frame ID to 0 in the request.



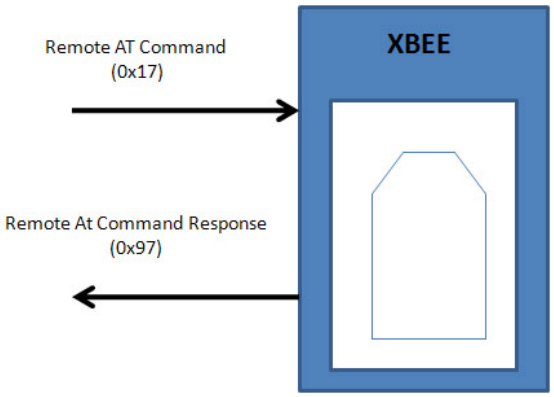
Transmitting and Receiving RF Data

The following image shows the API exchanges that take place at the serial port when sending RF data to another device. The transmit status frame is always sent at the end of a data transmission unless the frame ID is set to 0 in the transmit request. If the packet cannot be delivered to the destination, the transmit status frame will indicate the cause of failure. The received data frame (0x90 or 0x91) is set by the AP command.



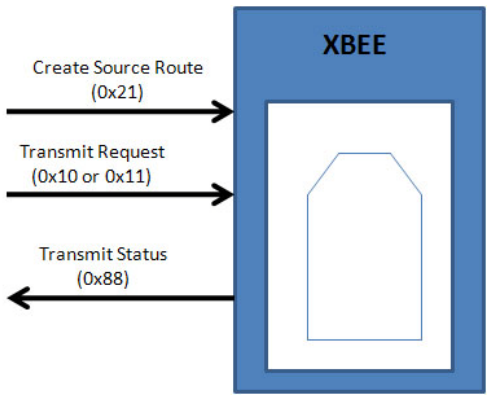
Remote AT Commands

The following image shows the API frame exchanges that take place at the serial port when sending a remote AT command. A remote command response frame is not sent out the serial port if the remote device does not receive the remote command.



Source Routing

The following image shows the API frame exchanges that take place at the serial port when sending a source routed transmission.



Supporting the API

Applications that support the API should make provisions to deal with new API frames that may be introduced in future releases. For example, a section of code on a host microprocessor that handles received serial API frames (sent out the module's DOUT pin) might look like this:

```
void XBee_HandleRxAPIFrame(_apiFrameUnion *papiFrame){
   switch(papiFrame->api_id){
      case RX_RF_DATA_FRAME:
         //process received RF data frame
         break;

      case RX_IO_SAMPLE_FRAME:
         //process IO sample frame
         break;

      case NODE_IDENTIFICATION_FRAME:
         //process node identification frame
         break;

      default:
         //Discard any other API frame types that are not being used
         break;
   }
}
```

API Frames

The following sections illustrate the types of frames encountered while using the API.

AT Command

Frame Type: 0x08
Used to query or set module parameters on the local device. This API command applies changes after executing the command. (Changes made to module parameters take effect once changes are applied.) The API example below illustrates an API frame when modifying the NJ parameter value of the module

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x04	
	Frame-specific Data	Frame Type	3	0x08	
		Frame ID	4	0x52 (R)	Identifies the serial port data frame for the host to correlate with a subsequent ACK (acknowledgement). If set to 0, no response is sent.
		AT Command	5	0x4E (N)	Command Name - Two ASCII characters that identify the AT Command.
			6	0x4A (J)	
		Parameter Value (optional)			If present, indicates the requested parameter value to set the given register. If no characters present, register is queried.
	Checksum		7	0x0D	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

The above example illustrates an AT command when querying an NJ value.

AT Command - Queue Parameter Value

Frame Type: 0x09

This API type allows module parameters to be queried or set. In contrast to the “AT Command” API type, new parameter values are queued and not applied until either the “AT Command” (0x08) API type or the AC (Apply Changes) command is issued. Register queries (reading parameter values) are returned immediately.

Example: Send a command to change the baud rate (BD) to 115200 baud, but don't apply changes yet. (Module will continue to operate at the previous baud rate until changes are applied.)

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x05	
	Frame-specific Data	Frame Type	3	0x09	
		Frame ID	4	0x01	Identifies the serial port data frame for the host to correlate with a subsequent ACK (acknowledgement). If set to 0, no response is sent.
		AT Command	5	0x42 (B)	Command Name - Two ASCII characters that identify the AT Command.
			6	0x44 (D)	
		Parameter Value (ATBD7 = 115200 baud)	7	0x07	If present, indicates the requested parameter value to set the given register. If no characters present, register is queried.
	Checksum		8	0x68	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Note: In this example, the parameter could have been sent as a zero-padded 2-byte or 4-byte value.

ZigBee Transmit Request

Frame Type: 0x10

A Transmit Request API frame causes the module to send data as an RF packet to the specified destination.

The 64-bit destination address should be set to 0x000000000000FFFF for a broadcast transmission (to all devices). The coordinator can be addressed by either setting the 64-bit address to all 0x00s and the 16-bit address to 0xFFFE, OR by setting the 64-bit address to the coordinator's 64-bit address and the 16-bit address to 0x0000. For all other transmissions, setting the 16-bit address to the correct 16-bit address can help improve performance when transmitting to multiple destinations. If a 16-bit address is not known, this field should be set to 0xFFFE (unknown). The Transmit Status frame (0x8B) will indicate the discovered 16-bit address, if successful.

The broadcast radius can be set from 0 up to NH. If set to 0, the value of NH specifies the broadcast radius (recommended). This parameter is only used for broadcast transmissions.

The maximum number of payload bytes can be read with the NP command.

Note: if source routing is used, the RF payload will be reduced by two bytes per intermediate hop in the source route. This example shows if escaping is disabled (AP=1).

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x16	
	Frame-specific Data	Frame Type	3	0x10	
		Frame ID	4	0x01	Identifies the serial port data frame for the host to correlate with a subsequent ACK (acknowledgement). If set to 0, no response is sent.
		64-bit Destination Address	MSB 5	0x00	Set to the 64-bit address of the destination device. The following addresses are also supported: 0x0000000000000000 - Reserved 64-bit address for the coordinator 0x000000000000FFFF - Broadcast address
			6	0x13	
			7	0xA2	
			8	0x00	
			9	0x40	
			10	0x0A	
			11	0x01	
			LSB 12	0x27	
		16-bit Destination Network Address	MSB 13	0xFF	Set to the 16-bit address of the destination device, if known. Set to 0xFFFE if the address is unknown, or if sending a broadcast.
			LSB 14	0xFE	
		Broadcast Radius	15	0x00	Sets maximum number of hops a broadcast transmission can occur. If set to 0, the broadcast radius will be set to the maximum hops value.
		Options	16	0x00	Bitfield of supported transmission options. Supported values include the following: 0x01 - Disable retries 0x20 - Enable APS encryption (if EE=1) 0x40 - Use the extended transmission timeout for this destination Enabling APS encryption decreases the maximum number of RF payload bytes by 4 (below the value reported by NP). Setting the extended timeout bit causes the stack to set the extended transmission timeout for the destination address. (See chapter 4.) All unused and unsupported bits must be set to 0.
		RF Data	17	0x54	Data that is sent to the destination device
			18	0x78	
			19	0x44	
			20	0x61	
			21	0x74	
			22	0x61	
			23	0x30	
			24	0x41	
	Checksum		25	0x13	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Example: The example above shows how to send a transmission to a module where escaping is disabled (AP=1) with destination address 0x0013A200 40014011, payload "TxData1B". If escaping is enabled (AP=2), the frame should look like:

0x7E 0x00 0x16 0x10 0x01 0x00 0x7D 0x33 0xA2 0x00 0x40 0x0A 0x01 0x27

0xFF 0xFE 0x00 0x00 0x54 0x78 0x44 0x61 0x74 0x61 0x30 0x41 0x7D 0x33

The checksum is calculated (on all non-escaped bytes) as [0xFF - (sum of all bytes from API frame type through data payload)].

Example: Send a transmission to the coordinator without specifying the coordinator's 64-bit address. The API transmit request frame should look like:

0x7E 0x00 0x16 0x10 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0xFF 0xFE 0x00 0x00 0x54 0x78 032 0x43 0x6F 0x6F 0x72 0x64 0xFC

Where 0x16 = length (22 bytes excluding checksum)

0x10 = ZigBee Transmit Request API frame type

0x01 = Frame ID (set to non-zero value)

0x0000000000000000 = Coordinator's address (can be replaced with coordinator's actual 64-bit address if known

0xFFFE = 16-bit Destination Address

0x00 = Broadcast radius

0x00 = Options

0x547832436F6F7264 = Data payload ("Tx2Coord")

0xFC = Checksum

Explicit Addressing ZigBee Command Frame

Frame Type: 0x11

Allows ZigBee application layer fields (endpoint and cluster ID) to be specified for a data transmission.

Similar to the ZigBee Transmit Request, but also requires ZigBee application layer addressing fields to be specified (endpoints, cluster ID, profile ID). An Explicit Addressing Request API frame causes the module to send data as an RF packet to the specified destination, using the specified source and destination endpoints, cluster ID, and profile ID.

The 64-bit destination address should be set to 0x000000000000FFFF for a broadcast transmission (to all devices). The coordinator can be addressed by either setting the 64-bit address to all 0x00s and the 16-bit address to 0xFFFE, OR by setting the 64-bit address to the coordinator's 64-bit address and the 16-bit address to 0x0000. For all other transmissions, setting the 16-bit address to the correct 16-bit address can help improve performance when transmitting to multiple destinations. If a 16-bit address is not known, this field should be set to 0xFFFE (unknown). The Transmit Status frame (0x8B) will indicate the discovered 16-bit address, if successful.

The broadcast radius can be set from 0 up to NH. If set to 0, the value of NH specifies the broadcast radius (recommended). This parameter is only used for broadcast transmissions.

The maximum number of payload bytes can be read with the NP command. Note: if source routing is used, the RF payload will be reduced by two bytes per intermediate hop in the source route.

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x1A	
	Frame-specific Data	Frame Type	3	0x11	
		Frame ID	4	0x01	Identifies the serial port data frame for the host to correlate with a subsequent ACK (acknowledgement). If set to 0, no response is sent.
		64-bit Destination Address	MSB 5	0x00	Set to the 64-bit address of the destination device. The following addresses are also supported: 0x0000000000000000 - Reserved 64-bit address for the coordinator 0x000000000000FFFF - Broadcast address
			6	0x00	
			7	0x00	
			8	0x00	
			9	0x00	
			10	0x00	
			11	0x00	
			12	0x00	
		16-bit Destination Network Address	MSB 13	0xFF	Set to the 16-bit address of the destination device, if known. Set to 0xFFFE if the address is unknown, or if sending a broadcast.
			LSB 14	0xFE	
		Source Endpoint	15	0xA0	Source endpoint for the transmission.
		Destination Endpoint	16	0xA1	Destination endpoint for the transmission.
		Cluster ID	17	0x15	Cluster ID used in the transmission
			18	0x54	
		Profile ID	19	0xC1	Profile ID used in the transmission
			20	0x05	
		Broadcast Radius	21	0x00	Sets the maximum number of hops a broadcast transmission can traverse. If set to 0, the transmission radius will be set to the network maximum hops value.
		Transmit Options	22	0x00	Bitfield of supported transmission options. Supported values include the following: 0x01 - Disable retries 0x04- Indirect Addressing 0x08- Multicast Addressing 0x20 - Enable APS encryption (if EE=1) 0x40 - Use the extended transmission timeout for this destination Enabling APS encryption decreases the maximum number of RF payload bytes by 4 (below the value reported by NP). Setting the extended timeout bit causes the stack to set the extended transmission timeout for the destination address. (See chapter 4.) All unused and unsupported bits must be set to 0.
		Data Payload	23	0x54	Data that is sent to the destination device
			24	0x78	
			25	0x44	
			26	0x61	
			27	0x74	
			28	0x61	
	Checksum		29	0x3A	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Example: Send a data transmission to the coordinator (64-bit address of 0x00s) using a source endpoint of 0xA0, destination endpoint 0xA1, cluster ID =0x1554, and profile ID 0xC105. Payload will be "TxData".

Remote AT Command Request

Frame Type: 0x17

Used to query or set module parameters on a remote device. For parameter changes on the remote device to take effect, changes must be applied, either by setting the apply changes options bit, or by sending an AC command to the remote.

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x10	
	Frame-specific Data	Frame Type	3	0x17	
		Frame ID	4	0x01	Identifies the serial port data frame for the host to correlate with a subsequent ACK (acknowledgement). If set to 0, no response is sent.
		64-bit Destination Address	MSB 5	0x00	Set to the 64-bit address of the destination device. The following addresses are also supported: 0x0000000000000000 - Reserved 64-bit address for the coordinator 0x000000000000FFFF - Broadcast address
			6	0x13	
			7	0xA2	
			8	0x00	
			9	0x40	
			10	0x40	
			11	0x11	
			LSB 12	0x22	
		16-bit Destination Network Address	MSB 13	0xFF	Set to the 16-bit address of the destination device, if known. Set to 0xFFFE if the address is unknown, or if sending a broadcast.
			LSB 14	0xFE	
		Remote Command Options	15	0x02 (apply changes)	Bitfield to enable various remote command options. Supported values include: 0x01 - Disable ACK 0x02 - Apply changes on remote. (If not set, AC command must be sent before changes will take effect.) 0x40 - Use the extended transmission timeout for this destination. Setting the extended timeout bit causes the stack to set the extended transmission timeout for the destination address (see chapter 4). All unused and unsupported bits must be set to 0.
		AT Command	16	0x42 (B)	Name of the command
			17	0x48 (H)	
		Command Parameter	18	0x01	If present, indicates the requested parameter value to set the given register. If no characters present, the register is queried.
	Checksum		19	0xF5	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Example: Send a remote command to change the broadcast hops register on a remote device to 1 (broadcasts go to 1-hop neighbors only), and apply changes so the new configuration value immediately takes effect. In this example, the 64-bit address of the remote is 0x0013A200 40401122, and the destination 16-bit address is unknown.

Create Source Route

Frame Type: 0x21

This frame creates a source route in the module. A source route specifies the complete route a packet should traverse to get from source to destination. Source routing should be used with many-to-one routing for best results.

Note: Both the 64-bit and 16-bit destination addresses are required when creating a source route. These are obtained when a Route Record Indicator (0xA1) frame is received.

Frame Fields		Offset	Example	Description
Start Delimiter		0	0x7E	
Length		MSB 1	0x00	Number of bytes between the length and the checksum
		LSB 2	0x14	
Frame-specific Data	Frame Type	3	0x21	
	Frame ID	4	0x00	The Frame ID should always be set to 0.
	64-bit Destination Address	MSB 5	0x00	Set to the 64-bit address of the destination device. The following addresses are also supported: 0x0000000000000000 - Reserved 64-bit address for the coordinator 0x000000000000FFFF - Broadcast address
		6	0x13	
		7	0xA2	
		8	0x00	
		9	0x40	
		10	0x40	
		11	0x11	
		LSB 12	0x22	
	16-bit Destination Network Address	MSB 13	0x33	Set to the 16-bit address of the destination device, if known. Set to 0xFFFE if the address is unknown, or if sending a broadcast.
		LSB 14	0x44	
	Route Command Options	15	0x00	Set to 0.
	Number of Addresses	16	0x03	The number of addresses in the source route (excluding source and destination). If this number is 0 or greater than the source route table size (40), this API frame will be silently discarded. However, there is no use in including more than 11 intermediate hops because a frame with more hops than that will be discarded.
	Address 1	17	0xEE	(neighbor of destination)
		18	0xFF	
	Address 2 (closer hop	19	0xCC	Address of intermediate hop
		20	0xDD	
	Address 3	21	0xAA	(neighbor of source)
		22	0xBB	
Checksum		23	0x01	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Example: Intermediate hop addresses must be ordered starting with the neighbor of the destination, and working closer to the source. For example, suppose a route is found between A and E as shown below.

A ' B ' C ' D ' E

If device E has the 64-bit and 16-bit addresses of 0x0013A200 40401122 and 0x3344, and if devices B, C, and D have the following 16-bit addresses:

B = 0xAABB

C = 0xCCDD

D = 0xEEFF

The example above shows how to send the Create Source Route frame to establish a source route between A and E.

AT Command Response

Frame Type: 0x88

In response to an AT Command message, the module will send an AT Command Response message. Some commands will send back multiple frames (for example, the ND (Node Discover) command).

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x05	
	Frame-specific Data	Frame Type	3	0x88	
		Frame ID	4	0x01	Identifies the serial port data frame being reported. Note: If Frame ID = 0 in AT Command Mode, no AT Command Response will be given.
		AT Command	5	'B' = 0x42	Command Name - Two ASCII characters that identify the AT Command.
			6	'D' = 0x44	
		Command Status	7	0x00	0 = OK 1 = ERROR 2 = Invalid Command 3 = Invalid Parameter 4 = Tx Failure
		Command Data			Register data in binary format. If the register was set, then this field is not returned, as in this example.
	Checksum		8	0xF0	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Example: Suppose the BD parameter is changed on the local device with a frame ID of 0x01. If successful (parameter was valid), the above response would be received.

Modem Status

Frame Type: (0x8A)

RF module status messages are sent from the module in response to specific conditions.

Example: The following API frame is returned when an API coordinator forms a network.

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x02	
	Frame-specific Data	Frame Type	3	0x8A	
		Status	4	0x06	0 = Hardware reset 1 = Watchdog timer reset 2 =Joined network (routers and end devices) 3 =Disassociated 6 =Coordinator started 7 = Network security key was updated 0x0D = Voltage supply limit exceeded (PRO only) 0x11 = Modem configuration changed while join in progress 0x80+ = Ember ZigBee stack error
	Checksum		5	0x6F	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Note: New modem status codes may be added in future firmware releases.

ZigBee Transmit Status

Frame Type: 0x8B

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.

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x07	
	Frame-specific Data	Frame Type	3	0x8B	
		Frame ID	4	0x01	Identifies the serial port data frame being reported. Note: If Frame ID = 0 in AT Command Mode, no AT Command Response will be given.
		16-bit address of destination	5	0x7D	16-bit Network Address the packet was delivered to (if successful). If not successful, this address will be 0xFFFD: Destination Address Unknown.
			6	0x84	
		Transmit Retry Count	7	0x00	The number of application transmission retries that took place.
		Delivery Status	8	0x00	0x00 = Success 0x01 = MAC ACK Failure 0x02 = CCA Failure 0x15 = Invalid destination endpoint 0x21 = Network ACK Failure 0x22 = Not Joined to Network 0x23 = Self-addressed 0x24 = Address Not Found 0x25 = Route Not Found 0x26 = Broadcast source failed to hear a neighbor relay the message 0x2B = Invalid binding table index 0x2C = Resource error lack of free buffers, timers, etc. 0x2D = Attempted broadcast with APS transmission 0x2E = Attempted unicast with APS transmission, but EE=0 0x32 = Resource error lack of free buffers, timers, etc. 0x74 = Data payload too large 0x75 = Indirect message unrequested
		Discovery Status	9	0x01	0x00 = No Discovery Overhead 0x01 = Address Discovery 0x02 = Route Discovery 0x03 = Address and Route 0x40 = Extended Timeout Discovery
	Checksum		10	0x71	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Example: Suppose a unicast data transmission was sent to a destination device with a 16-bit address of 0x7D84. (The transmission could have been sent with the 16-bit address set to 0x7D84 or 0xFFFE.)

ZigBee Receive Packet

Frame Type: (0x90)

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

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x11	
	Frame-specific Data	Frame Type	3	0x90	
		64-bit Source Address	MSB 4	0x00	
			5	0x13	64-bit address of sender. Set to 0xFFFFFFFFFFFFFFFF (unknown 64-bit address) if the sender's 64-bit address is unknown.
			6	0xA2	
			7	0x00	
			8	0x40	
			9	0x52	
			10	0x2B	
			LSB 11	0xAA	
		16-bit Source Network Address	MSB 12	0x7D	16-bit address of sender
			LSB 13	0x84	
		Receive Options	14	0x01	0x01 - Packet was sent with retries disabled 0x02 - Packet was a broadcast packet 0x20 - Packet encrypted with APS encryption 0x40 - Packet was sent from an end device (if known)
		Received Data	15	0x52	Received RF data
			16	0x78	
			17	0x44	
			18	0x61	
			19	0x74	
			20	0x61	
	Checksum		21	0x0D	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Example: Suppose a device with a 64-bit address of 0x0013A200 40522BAA, and 16-bit address 0x7D84 sends a unicast data transmission to a remote device with payload "RxData". If AO=0 on the receiving device, it would send the above example frame out its serial port.

ZigBee Explicit Rx Indicator

Frame Type:0x91

When the modem receives a ZigBee RF packet it is sent out the serial port using this message type (when AO=1).

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x18	
	Frame-specific Data	Frame Type	3	0x91	
		64-bit Source Address	MSB 4	0x00	64-bit address of sender. Set to 0xFFFFFFFFFFFFFFFF (unknown 64-bit address) if the sender's 64-bit address is unknown.
			5	0x13	
			6	0xA2	
			7	0x00	
			8	0x40	
			9	0x52	
			10	0x2B	
			LSB 11	0xAA	
		16-bit Source Network Address	MSB 12	0x7D	16-bit address of sender.
			LSB 13	0x84	
		Source Endpoint	14	0xE0	Endpoint of the source that initiated the transmission
		Destination Endpoint	15	0xE0	Endpoint of the destination the message is addressed to.
		Cluster ID	16	0x22	Cluster ID the packet was addressed to.
			17	0x11	
		Profile ID	18	0xC1	Profile ID the packet was addressed to.
			19	0x05	
		Receive Options	20	0x02	0x01 – Packet Acknowledged 0x02 – Packet was a broadcast packet 0x20 - Packet encrypted with APS encryption 0x40 - Packet was sent from an end device (if known)
		Received Data	21	0x52	Received RF data
			22	0x78	
			23	0x44	
			24	0x61	
			25	0x74	
			26	0x61	
	Checksum		27	0x52	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Example: Suppose a device with a 64-bit address of 0x0013A200 40522BAA, and 16-bit address 0x7D84 sends a broadcast data transmission to a remote device with payload "RxData". Suppose the transmission was sent with source and destination endpoints of 0xE0, cluster ID=0x2211, and profile ID=0xC105. If AO=1 on the receiving device, it would send the above frame out its serial port.

ZigBee IO Data Sample Rx Indicator

Frame Type: 0x92

When the module receives an I/O sample frame from a remote device, it sends the sample out the serial port using this frame type (when AO=0). Only modules running in API mode will send I/O samples out the serial port.

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x14	
	Frame-specific Data	Frame Type	3	0x92	
		64-bit Source Address	MSB 4	0x00	64-bit address of sender
			5	0x13	
			6	0xA2	
			7	0x00	
			8	0x40	
			9	0x52	
			10	0x2B	
			LSB 11	0xAA	
		16-bit Source Network Address	MSB 12	0x7D	16-bit address of sender.
			LSB 13	0x84	
		Receive Options	14	0x01	0x01 - Packet Acknowledged 0x02 - Packet was a broadcast packet
		Number of Samples	15	0x01	Number of sample sets included in the payload. (Always set to 1)
		Digital Channel Mask*	16	0x00	Bitmask field that indicates which digital IO lines on the remote have sampling enabled (if any).
			17	0x1C	
		Analog Channel Mask**	18	0x02	Bitmask field that indicates which analog IO lines on the remote have sampling enabled (if any).
		Digital Samples (if included)	19	0x00	If the sample set includes any digital IO lines (Digital Channel Mask > 0), these two bytes contain samples for all enabled digital IO lines. DIO lines that do not have sampling enabled return 0. Bits in these 2 bytes map the same as they do in the Digital Channels Mask field.
			20	0x14	
		Analog Sample	21	0x02	If the sample set includes any analog input lines (Analog Channel Mask > 0), each enabled analog input returns a 2-byte value indicating the A/D measurement of that input. Analog samples are ordered sequentially from AD0/DIO0 to AD3/DIO3, to the supply voltage.
			22	0x25	
	Checksum		23	0xF5	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

*	N/A	N/A	N/A	CD/DIO12	PWM/DIO11	RSSI/DIO10	N/A	N/A
	CTS/DIO7	RTS/DIO6	ASSOC/DIO5	DIO4	AD3/DIO3	AD2/DIO2	AD1/DIO1	AD0/DIO0
**	Supply Voltage	N/A	N/A	N/A	AD3	AD2	AD1	AD0

Example: Suppose an IO sample is received with analog and digital IO, from a remote with a 64-bit address of 0x0013A200 40522BAA and a 16-bit address of 0x7D84. If pin AD1/DIO1 is enabled as an analog input, AD2/DIO2 and DIO4 are enabled as a digital inputs (currently high), and AD3/DIO3 is enabled as a digital output (low) the IO sample is shown in the API example in the table above.

XBee Sensor Read Indicator

Frame Type: 0x94

When the module receives a sensor sample (from a Digi 1-wire sensor adapter), it is sent out the serial port using this message type (when AO=0).

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x17	
	Frame-specific Data	Frame Type	3	0x94	
		64-bit Source Address	MSB 4	0x00	64-bit address of sender
			5	0x13	
			6	0xA2	
			7	0x00	
			8	0x40	
			9	0x52	
			10	0x2B	
			LSB 11	0xAA	
		16-bit Source Network Address	MSB 12	0xDD	16-bit address of sender.
			LSB 13	0x6C	
		Receive Options	14	0x01	0x01 - Packet Acknowledged 0x02 - Packet was a broadcast packet
		1-Wire Sensors	15	0x03	0x01 = A/D Sensor Read 0x02 = Temperature Sensor Read 0x60 = Water present (module CD pin low)
		A/D Values	16	0x00	Indicates a two-byte value for each of four A/D sensors (A, B, C, D) Set to 0xFFFFFFFFFFFFFFFF if no A/Ds are found.
			17	0x02	
			18	0x00	
			19	0xCE	
			20	0x00	
			21	0xEA	
			22	0x00	
			23	0x52	
		Temperature Read	24	0x01	Indicates the two-byte value read from a digital thermometer if present. Set to 0xFFFF if not found.
			25	0x6A	
	Checksum		26	0x8B	0xFF - the 0x8 bit sum of bytes from offset 3 to this byte.

Example: Suppose a 1-wire sensor sample is received from a device with a 64-bit address of 0x0013A200 40522BAA and a 16-bit address of 0xDD6C. If the sensor sample was taken from a 1-wire humidity sensor, the API frame could look like this (if AO=0):

For convenience, let's label the A/D and temperature readings as AD0, AD1, AD2, AD3, and T. Using the data in this example:

AD0 = 0x0002

AD1 = 0x00CE

AD2 = 0x00EA

AD3 = 0x0052

T = 0x016A

To convert these to temperature and humidity values, the following equations should be used.

Temperature (°C) = (T / 16), for T < 2048

= - (T & 0x7FF) / 16, for T >= 2048

Vsupply = (AD2 * 5.1) / 255

Voutput = (AD3 * 5.1) / 255

Relative Humidity = ((Voutput / Vsupply) - 0.16) / (0.0062)

True Humidity = Relative Humidity / (1.0546 - (0.00216 * Temperature (°C)))

Looking at the sample data, we have:

Vsupply = (234 * 5.1 / 255) = 4.68

Voutput = (82 * 5.1 / 255) = 1.64

Temperature = (362 / 16) = 22.625°C

Relative H = (161.2903 * ((1.64/4.68) - 0.16)) = 161.2903 * (0.19043) = 30.71%

True H = (30.71 / (1.0546 - (0.00216 * 22.625))) = (30.71 / 1.00573) = 30.54%

Node Identification Indicator

Frame Type: 0x95

This frame is received when a module transmits a node identification message to identify itself (when AO=0). The data portion of this frame is similar to a network discovery response frame (see ND command).

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x20	
		Frame Type	3	0x95	
		64-bit Source Address	MSB 4	0x00	64-bit address of sender
			5	0x13	
			6	0xA2	
			7	0x00	
			8	0x40	
			9	0x52	
			10	0x2B	
			LSB 11	0xAA	
		16-bit Source Network Address	MSB 12	0x7D	16-bit address of sender.
			LSB 13	0x84	
	Frame-specific Data	Receive Options	14	0x02	0x01 - Packet Acknowledged 0x02 - Packet was a broadcast packet
		Source 16-bit address	15	0x7D	Set to the 16-bit network address of the remote. Set to 0xFFFE if unknown.
			16	0x84	
		64-bit Network address	17	0x00	Indicates the 64-bit address of the remote module that transmitted the node identification frame.
			18	0x13	
			19	0xA2	
			20	0x00	
			21	0x40	
			22	0x52	
			23	0x2B	
			24	0xAA	
		NI String	25	0x20	Node identifier string on the remote device. The NI-String is terminated with a NULL byte (0x00).
			26	0x00	
		Parent 16-bit address	27	0xFF	Indicates the 16-bit address of the remote's parent or 0xFFFE if the remote has no parent.
			28	0xFE	
		Device Type	29	0x01	0 = Coordinator 1 = Router 2 = End Device
		Source Event	30	0x01	1 = Frame sent by node identification pushbutton event (see D0 command) 2 = Frame sent after joining event occurred (see JN command). 3 = Frame sent after power cycle event occurred (see JN command).
		Digi Profile ID	31	0xC1	Set to Digi's application profile ID.
			32	0x05	
		Manufacturer ID	33	0x10	Set to Digi's Manufacturer ID.
			34	0x1E	
	Checksum		35	0x1B	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Example: If the commissioning push button is pressed on a remote router device with 64-bit address 0x0013A200 40522BAA, 16-bit address 0x7D84, and default NI string, the following node identification indicator would be received.

Remote Command Response

Frame Type: 0x97

If a module receives a remote command response RF data frame in response to a Remote AT Command Request, the module will send a Remote AT Command Response message out the serial port. Some commands may send back multiple frames--for example, Node Discover (ND) command.

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x13	
	Frame-specific Data	Frame Type	3	0x97	
		Frame ID	4	0x55	This is the same value passed in to the request. .
		64-bit Source (remote) Address	MSB 5	0x00	The address of the remote radio returning this response.
			6	0x13	
			7	0xA2	
			8	0x00	
			9	0x40	
			10	0x52	
			11	0x2B	
			LSB 12	0xAA	
		16-bit Source (remote) Address	MSB 13	0x7D	Set to the 16-bit network address of the remote. Set to 0xFFFE if unknown.
			LSB 14	0x84	
		AT Commands	15	0x53	Name of the command
			16	0x4C	
		Command Status	17	0x00	0 = OK 1 = ERROR 2 = Invalid Command 3 = Invalid Parameter 4 = Remote Command Transmission Failed
		Command Data	18	0x40	Register data in binary format. If the register was set, then this field is not returned.
			19	0x52	
			20	0x2B	
			21	0xAA	
	Checksum		22	0xF0	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Example: If a remote command is sent to a remote device with 64-bit address 0x0013A200 40522BAA and 16-bit address 0x7D84 to query the SL command, and if the frame ID=0x55, the response is shown in the example API frame in the table above.

Over-the-Air Firmware Update Status

Frame Type: 0xA0

The Over-the-Air Firmware Update Status frame provides a status indication of a firmware update transmission attempt. If a query command (0x01 0x51) is sent to a target with a 64-bit address of 0x0013A200 40522BAA through an updater with 64-bit address 0x0013A200403E0750 and 16-bit address 0x0000, the following is the expected response.

API Packet	Frame Fields		Offset	Example	Description
	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x16	
	Frame-specific Data	Frame Type	3	0xA0	
		64-bit Source (remote) Address	MSB 4	0x00	The address of the remote radio returning this response.
			5	0x13	
			6	0xA2	
			7	0x00	
			8	0x40	
			9	0x3E	
			10	0x07	
			LSB 11	0x50	
		16-bit Destination Address	12	0x00	16-bit address of the updater device
			13	0x00	
		Receive Options	14	0x01	0x01 - Packet Acknowledged. 0x02 - Packet was a broadcast.
		Bootloader Message Type	15	0x52	0x06 - ACK 0x15 - NACK 0x40 - No Mac ACK 0x51 - Query (received if the bootloader is not active on the target) 0x52 - Query Response
		Block Number	16	0x00	Block number used in the update request. Set to 0 if not applicable.
		64-bit Target Address	17	0x00	64-bit Address of remote device that is being updated (target).
			18	0x13	
			19	0xA2	
			20	0x00	
			21	0x40	
			22	0x52	
			23	0x2B	
			24	0xAA	
	Checksum		25	0x66	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

If a query request returns a 0x15 (NACK) status, the target is likely waiting for a firmware update image. If no messages are sent to it for about 75 seconds, the target will timeout and accept new query messages.

If a query returns a 0x51 (QUERY) status, then the target's bootloader is not active and will not respond to query messages.

Route Record Indicator

Frame Type: 0xA1

The route record indicator is received whenever a device sends a ZigBee route record command. This is used with many-to-one routing to create source routes for devices in a network.

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x13	
	Frame-specific Data	Frame Type	3	0xA1	
		64-bit Source Address	MSB 4	0x00	64-bit address of the device that initiated the route record.
			5	0x13	
			6	0xA2	
			7	0x00	
			8	0x40	
			9	0x40	
			10	0x11	
			LSB 11	0x22	
		Source (updater) 16-bit Address	12	0x33	16-bit address of the device that initiated the route record.
			13	0x44	
		Receive Options	14	0x01	0x01 - Packet Acknowledged. 0x02 - Packet was a broadcast.
		Number of Addresses	15	0x03	The number of addresses in the source route (excluding source and destination).
		Address 1	16	0xEE	(neighbor of destination)
			17	0xFF	
		Address 2 (closer hop	18	0xCC	Address of intermediate hop
			19	0xDD	
		Address n (neighbor of source)	20	0xAA	Two bytes per 16-bit address.
			21	0xBB	
	Checksum		22	0x80	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Example: Suppose device E sends a route record that traverses multiple hops en route to data collector device A as shown below.

A B C D E

If device E has the 64-bit and 16-bit addresses of 0x0013A200 40401122 and 0x3344, and if devices B, C, and D have the following 16-bit addresses:

B = 0xAABB

C = 0xCCDD

D = 0xEEFF

The data collector will send the above API frame out its serial port.

Many-to-One Route Request Indicator

Frame Type: 0xA3

The many-to-one route request indicator frame is sent out the serial port whenever a many-to-one route request is received

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x0C	
	Frame-specific Data	Frame Type	3	0xA3	
		64-bit Source Address	MSB 4	0x00	64-bit address of the device that sent the many-to-one route request
			5	0x13	
			6	0xA2	
			7	0x00	
			8	0x40	
			9	0x40	
			10	0x11	
			LSB 11	0x22	
		Source 16-bit Address	MSB 12	0x00	16-bit address of the device that initiated the many-to-one route request.
			LSB 13	0x00	
		Reserved	14	0x00	Set to 0.
	Checksum		15	0xF4	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Example: Suppose a device with a 64-bit address of 0x0013A200 40401122 and 16-bit address of 0x0000 sends a many-to-one route request. All remote routers operating in API mode that receive the many-to-one broadcast would send the above example API frame out their serial port.

Sending ZigBee Device Objects (ZDO) Commands with the API

ZigBee Device Objects (ZDOs) are defined in the ZigBee Specification as part of the ZigBee Device Profile. These objects provide functionality to manage and map out the ZigBee network and to discover services on ZigBee devices. ZDOs are typically required when developing a ZigBee product that will interoperate in a public profile such as home automation or smart energy, or when communicating with ZigBee devices from other vendors. The ZDO can also be used to perform several management functions such as frequency agility (energy detect and channel changes - Mgmt Network Update Request), discovering routes (Mgmt Routing Request) and neighbors (Mgmt LQI Request), and managing device connectivity (Mgmt Leave and Permit Join Request).

The following table shows some of the more prominent ZDOs with their respective cluster identifier. Each ZDO command has a defined payload. See the "ZigBee Device Profile" section of the ZigBee Specification for details.

ZDO Command	Cluster ID
Network Address Request	0x0000
IEEE Address Request	0x0001
Node Descriptor Request	0x0002
Simple Descriptor Request	0x0004
Active Endpoints Request	0x0005
Match Descriptor Request	0x0006
Mgmt LQI Request	0x0031
Mgmt Routing Request	0x0032
Mgmt Leave Request	0x0034
Mgmt Permit Joining Request	0x0036
Mgmt Network Update Request	0x0038

The Explicit Transmit API frame (0x11) is used to send ZigBee Device Objects commands to devices in the network. Sending ZDO commands with the Explicit Transmit API frame requires some formatting of the data payload field.

When sending a ZDO command with the API, all multiple byte values in the ZDO command (API payload) (e.g. u16, u32, 64-bit addresses) must be sent in little endian byte order for the command to be executed correctly on a remote device.

For an API XBee to receive ZDO responses, the AO command must be set to 1 to enable the explicit receive API frame.

The following table shows how the Explicit API frame can be used to send an "Active Endpoints" request to discover the active endpoints on a device with a 16-bit address of 0x1234.

	Frame Fields			Offset	Example	Description
API Packet	Start Delimiter			0	0x7E	
	Length			MSB 1	0x00	Number of bytes between the length and the checksum
				LSB 2	0x17	
	Frame-specific Data	Frame Type		3	0x11	
		Frame ID		4	0x01	Identifies the serial port data frame for the host to correlate with a subsequent transmit status. If set to 0, no transmit status frame will be sent out the serial port.
		64-bit Destination Address		MSB 5	0x00	64-bit address of the destination device (big endian byte order). For unicast transmissions, set to the 64-bit address of the destination device, or to 0x0000000000000000 to send a unicast to the coordinator. Set to 0x000000000000FFFF for broadcast.
				6	0x00	
				7	0x00	
				8	0x00	
				9	0x00	
				10	0x00	
				11	0xFF	
				12	0xFF	
		16-bit Destination Network Address		MSB 13	0xFF	16-bit address of the destination device (big endian byte order). Set to 0xFFFE for broadcast, or if the 16-bit address is unknown.
				LSB 14	0xFE	
		Source Endpoint		15	0x00	Set to 0x00 for ZDO transmissions (endpoint 0 is the ZDO endpoint).
		Destination Endpoint		16	0x00	Set to 0x00 for ZDO transmissions (endpoint 0 is the ZDO endpoint).
		Cluster ID		MSB 17	0x00	Set to the cluster ID that corresponds to the ZDO command being sent. 0x0005 = Active Endpoints Request
				LSB 18	0x05	
		Profile ID		MSB 19	0x00	Set to 0x0000 for ZDO transmissions (Profile ID 0x0000 is the ZigBee Device Profile that supports ZDOs).
				LSB 20	0x00	
		Broadcast Radius		21	0x00	Sets the maximum number of hops a broadcast transmission can traverse. If set to 0, the transmission radius will be set to the network maximum hops value.
		Transmit Options		22	0x00	All bits must be set to 0.
		Data Payload	Transaction Sequence Number	23	0x01	The required payload for a ZDO command. All multi-byte ZDO parameter values (u16, u32, 64-bit address) must be sent in little endian byte order. The Active Endpoints Request includes the following payload: [16-bit NwkAddrOfInterest] Note the 16-bit address in the API example (0x1234) is sent in little endian byte order (0x3412).
			ZDO Payload	24	0x34	
				25	0x12	
	Checksum			26	0xA6	0xFF minus the 8 bit sum of bytes from offset 3 to this byte.

Sending ZigBee Cluster Library (ZCL) Commands with the API

The ZigBee Cluster Library defines a set of attributes and commands (clusters) that can be supported in multiple ZigBee profiles. The ZCL commands are typically required when developing a ZigBee product that will interoperate in a public profile such as home automation or smart energy, or when communicating with ZigBee devices from other vendors. Applications that are not designed for a public profile or for interoperability applications can skip this section.

The following table shows some prominent clusters with their respective attributes and commands.

Cluster (Cluster ID)	Attributes (Attribute ID)	Cluster ID
Basic (0x0000)	Application Version (0x0001) Hardware Version (0x0003) Model Identifier (0x0005)	-Reset to defaults (0x00)
Identify (0x0003)	Identify Time (0x0000)	Identify (0x00) Identify Query (0x01)
Time (0x000A)	Time (0x0000) Time Status (0x0001) Time Zone (0x0002)	
Thermostat (0x0201)	Local Temperature (0x0000) Occupancy (0x0002)	-Setpoint raise / lower (0x00)

The ZCL defines a number of profile-wide commands that can be supported on any profile, also known as general commands. These commands include the following.

Command (Command ID)	Description
Read Attributes (0x00)	Used to read one or more attributes on a remote device.
Read Attributes Response (0x01)	Generated in response to a read attributes command.
Write Attributes (0x02)	Used to change one or more attributes on a remote device.
Write Attributes Response (0x04)	Sent in response to a write attributes command.
Configure Reporting (0x06)	Used to configure a device to automatically report on the values of one or more of its attributes.
Report Attributes (0x0A)	Used to report attributes when report conditions have been satisfied.
Discover Attributes (0x0C)	Used to discover the attribute identifiers on a remote device.
Discover Attributes Response (0x0D)	Sent in response to a discover attributes command.

The Explicit Transmit API frame (0x11) is used to send ZCL commands to devices in the network. Sending ZCL commands with the Explicit Transmit API frame requires some formatting of the data payload field.

When sending a ZCL command with the API, all multiple byte values in the ZCL command (API Payload) (e.g. u16, u32, 64-bit addresses) must be sent in little endian byte order for the command to be executed correctly on a remote device.

Note: When sending ZCL commands, the AO command should be set to 1 to enable the explicit receive API frame. This will provide indication of the source 64- and 16-bit addresses, cluster ID, profile ID, and endpoint information for each received packet. This information is required to properly decode received data.

The following table shows how the Explicit API frame can be used to read the hardware version attribute from a device with a 64-bit address of 0x0013A200 40401234 (unknown 16-bit address). This example uses arbitrary source and destination endpoints. Recall the hardware version attribute (attribute ID 0x0003) is part of the basic cluster (cluster ID 0x0000). The Read Attribute general command ID is 0x00.

	Frame Fields			Offset	Example	Description
API Packet	Start Delimiter			0	0x7E	
	Length			MSB 1	0x00	Number of bytes between the length and the checksum
				LSB 2	0x19	
	Frame-specific Data	Frame Type		3	0x11	
		Frame ID		4	0x01	Identifies the serial port data frame for the host to correlate with a subsequent transmit status. If set to 0, no transmit status frame will be sent out the serial port.
		64-bit Destination Address		MSB 5	0x00	64-bit address of the destination device (big endian byte order). For unicast transmissions, set to the 64-bit address of the destination device, or to 0x0000000000000000 to send a unicast to the coordinator. Set to 0x000000000000FFFF for broadcast.
				6	0x13	
				7	0xA2	
				8	0x00	
				9	0x40	
				10	0x40	
				11	0x12	
				12	0x34	
		16-bit Destination Network Address		MSB 13	0xFF	16-bit address of the destination device (big endian byte order). Set to 0xFFFE for broadcast, or if the 16-bit address is unknown.
				LSB 14	0xFE	
		Source Endpoint		15	0x41	Set to the source endpoint on the sending device. (0x41 arbitrarily selected).
		Destination Endpoint		16	0x42	Set to the destination endpoint on the remote device. (0x42 arbitrarily selected)
		Cluster ID		MSB 17	0x00	Set to the cluster ID that corresponds to the ZCL command being sent. 0x0000 = Basic Cluster
				LSB 18	0x00	
		Profile ID		MSB 19	0xD1	Set to the profile ID supported on the device. (0xD123 arbitrarily selected).
				LSB 20	0x23	
		Broadcast Radius		21	0x00	Sets the maximum number of hops a broadcast transmission can traverse. If set to 0, the transmission radius will be set to the network maximum hops value.
		Transmit Options		22	0x00	All bits must be set to 0.
		Data Payload	Frame Control	23	0x00	Bitfield that defines the command type and other relevant information in the ZCL command. See the ZCL specification for details.
			ZCL Frame Header	Transaction Sequence Number	24	0x01	A sequence number used to correlate a ZCL command with a ZCL response. (The hardware version response will include this byte as a sequence number in the response.) The value 0x01 was arbitrarily selected.
				Command ID	25	0x00	Since the frame control "frame type" bits are 00, this byte specifies a general command. Command ID 0x00 is a Read Attributes command.
			ZCL Payload	Attribute ID	26	0x03	The payload for a "Read Attributes" command is a list of Attribute Identifiers that are being read. Note the 16-bit Attribute ID (0x0003) is sent in little endian byte order (0x0300). All multi-byte ZCL header and payload values must be sent in little endian byte order.
					27	0x00	
	Checksum				28	0xFA	0xFF minus the 8 bit sum of bytes from offset 3 to this byte.

In the above example, the Frame Control field (offset 23) was constructed as follows:

Name	Bits	Example Value Description
Frame Type	0-1	00 - Command acts across the entire profile
Manufacturer Specific	2	0 - The manufacturer code field is omitted from the ZCL Frame Header.
Direction	3	0 - The command is being sent from the client side to the server side.
Disable Default Response	4	0 - Default response not disabled
Reserved	5-7	Set to 0.

See the ZigBee Cluster Library specification for details.

Sending Public Profile Commands with the API

Commands in public profiles such as Smart Energy and Home Automation can be sent with the XBee API using the Explicit Transmit API frame (0x11). Sending public profile commands with the Explicit Transmit API frame requires some formatting of the data payload field. Most of the public profile commands fit into the ZigBee Cluster Library (ZCL) architecture as described in the previous section.

The following table shows how the Explicit API frame can be used to send a demand response and load control message (cluster ID 0x701) in the smart energy profile (profile ID 0x0109) in the revision 14 Smart Energy specification. The message will be a "Load Control Event" (command ID 0x00) and will be sent to a device with 64-bit address of 0x0013A200 40401234 with a 16-bit address of 0x5678. The event will start a load control event for water heaters and smart appliances, for a duration of 1 minute, starting immediately.

Note: When sending public profile commands, the AO command should be set to 1 to enable the explicit receive API frame. This will provide indication of the source 64- and 16-bit addresses, cluster ID, profile ID, and endpoint information for each received packet. This information is required to properly decode received data.

	Frame Fields				Offset	Example	Description
API Packet	Start Delimiter				0	0x7E	
	Length				MSB 1	0x00	Number of bytes between the length and the checksum
					LSB 2	0x19	
	Frame-specific Data	Frame Type			3	0x11	
		Frame ID			4	0x01	Identifies the serial port data frame for the host to correlate with a subsequent transmit status. If set to 0, no transmit status frame will be sent out the serial port.
		64-bit Destination Address			MSB 5	0x00	64-bit address of the destination device (big endian byte order). For unicast transmissions, set to the 64-bit address of the destination device, or to 0x0000000000000000 to send a unicast to the coordinator. Set to 0x000000000000FFFF for broadcast.
					6	0x13	
					7	0xA2	
					8	0x00	
					9	0x40	
					10	0x40	
					11	0x12	
					12	0x34	
		16-bit Destination Network Address			MSB 13	0x56	16-bit address of the destination device (big endian byte order). Set to 0xFFFE for broadcast, or if the 16-bit address is unknown.
					LSB 14	0x78	
		Source Endpoint			15	0x41	Set to the source endpoint on the sending device. (0x41 arbitrarily selected).
		Destination Endpoint			16	0x42	Set to the destination endpoint on the remote device. (0x42 arbitrarily selected)
		Cluster ID			MSB 17	0x07	Set to the cluster ID that corresponds to the ZCL command being sent. 0x0701 = Demand response and load control cluster ID
					LSB 18	0x01	
		Profile ID			MSB 19	0x01	Set to the profile ID supported on the device. 0x0109 = Smart Energy profile ID.
					LSB 20	0x09	
		Broadcast Radius			21	0x00	Sets the maximum number of hops a broadcast transmission can traverse. If set to 0, the transmission radius will be set to the network maximum hops value.
		Transmit Options			22	0x00	All bits must be set to 0.
		Data Payload	ZCL Frame Header	Frame Control	23	0x09	Bitfield that defines the command type and other relevant information in the ZCL command. See the ZCL specification for details.
				Transaction Sequence Number	24	0x01	A sequence number used to correlate a ZCL command with a ZCL response. (The hardware version response will include this byte as a sequence number in the response.) The value 0x01 was arbitrarily selected.
					25	0x00	Since the frame control "frame type" bits are 01, this byte specifies a cluster-specific command. Command ID 0x00 in the Demand Response and Load Control cluster is a Load Control Event command. (See Smart Energy specification.)

Frame Fields				Offset	Example	Description
			Issuer Event ID	26	0x78	4-byte unique identifier. Note the 4-byte ID is sent in little endian byte order (0x78563412). The event ID in this example (0x12345678) was arbitrarily selected.
				27	0x56	
				28	0x34	
				29	0x12	
			Device Class	30	0x14	to apply the load control event. A bit value of 0x0014 enables smart appliances and water heaters. Note the 2-byte bit field value is sent in little endian byte order.
				31	0x00	
			Utility Enrollment Group	32	0x00	Used to identify sub-groups of devices in the device-class. 0x00 addresses all groups.
			Start Time	33	0x00	UTC timestamp representing when the event should start. A value of 0x00000000 indicates "now".
				34	0x00	
				35	0x00	
				36	0x00	
			Duration in Minutes	37	0x01	This 2-byte value must be sent in little endian byte order.
				38	0x00	
		ZCL Payload - Load Control Event Data	Criticality Level	39	0x04	Indicates the criticality level of the event. In this example, the level is "voluntary".
			Cooling Temperature	40	0xFF	Requested offset to apply to the normal cooling set point. A value of 0xFF indicates the temperature offset value is not used.
			Heating Temperature Offset	41	0xFF	Requested offset to apply to the normal heating set point. A value of 0xFF indicates the temperature offset value is not used.
			Cooling Temperature Set Point	42	0x00	Requested cooling set point in 0.01 degrees Celsius. A value of 0x8000 means the set point field is not used in this event. Note the 0x80000 is sent in little endian byte order.
				43	0x80	
			Heating Temperature Set Point	44	0x00	Requested heating set point in 0.01 degrees Celsius. A value of 0x8000 means the set point field is not used in this event. Note the 0x80000 is sent in little endian byte order.
				45	0x80	
			Average Load Adjustment Percentage	46	0x80	Maximum energy usage limit. A value of 0x80 indicates the field is not used.
			Duty Cycle	47	0xFF	Defines the maximum "On" duty cycle. A value of 0xFF indicates the duty cycle is not used in this event.
			Duty Cycle Event Control	48	0x00	A bitmap describing event options.
Checksum				49	0x5B	0xFF minus the 8 bit sum of bytes from offset 3 to this byte.

In the above example, the Frame Control field (offset 23) was constructed as follows:

Name	Bits	Example Value Description
Frame Type	0-1	01 - Command is specific to a cluster
Manufacturer Specific	2	0 - The manufacturer code field is omitted from the ZCL Frame Header.
Direction	3	1 - The command is being sent from the server side to the client side.
Disable Default Response	4	0 - Default response not disabled
Reserved	5-7	Set to 0.

10. XBee Command Reference Tables

Addressing

Addressing Commands

AT Command	Name and Description	Parameter Range	Default
DH	Destination Address High.Set/Get the upper 32 bits of the 64-bit destination address. When combined with DL, it defines the 64-bit destination address for data transmission. Special definitions for DH and DL include 0x000000000000FFFF (broadcast) and 0x0000000000000000 (coordinator).	0 - 0xFFFFFFFF	0
DL	Destination Address Low. Set/Get the lower 32 bits of the 64-bit destination address. When combined with DH, it defines the 64-bit destination address for data transmissions. Special definitions for DH and DL include 0x000000000000FFFF (broadcast) and 0x0000000000000000 (coordinator).	0 - 0xFFFFFFFF	0xFFFF(Coordinator) 0 (Router/End Device)
MY	16-bit Network Address. Read the 16-bit network address of the module. A value of 0xFFFE means the module has not joined a ZigBee network	0 - 0xFFFE [read-only]	0xFFFE
MP	16-bit Parent Network Address. Read the 16-bit network address of the module's parent. A value of 0xFFFE means the module does not have a parent.	0 - 0xFFFE [read-only]	0xFFFE
NC	Number of Remaining Children. Read the number of end device children that can join the device. If NC returns 0, then the device cannot allow any more end device children to join.	0 - MAX_CHILDREN (maximum varies)	read-only
SH	Serial Number High. Read the high 32 bits of the module's unique 64-bit address.	0 - 0xFFFFFFFF [read-only]	factory-set
SL	Serial Number Low. Read the low 32 bits of the module's unique 64-bit address.	0 - 0xFFFFFFFF [read-only]	factory-set
NI	Node Identifier. Stores a string identifier. The register only accepts printable ASCII data. In AT Command Mode, a string can not start with a space. A carriage return ends the command. Command will automatically end when maximum bytes for the string have been entered. This string is returned as part of the ND (Node Discover) command. This identifier is also used with the DN (Destination Node) command. In AT command mode, an ASCII comma (0x2C) cannot be used in the NI string	20-Byte printable ASCII string	ASCII space character (0x20)
SE	Source Endpoint. Set/read the ZigBee application layer source endpoint value. This value will be used as the source endpoint for all data transmissions. SE is only used in transparent mode.The default value 0xE8 (Data endpoint) is the Digi data endpoint	0 - 0xFF	0xE8
DE	Destination Endpoint. Set/read Zigbee application layer destination ID value. This value will be used as the destination endpoint all data transmissions. DE is only used in transparent mode.The default value (0xE8) is the Digi data endpoint.	0 - 0xFF	0xE8
CI	Cluster Identifier. Set/read Zigbee application layer cluster ID value. This value will be used as the cluster ID for all data transmissions. CI is only used in transparent mode.The default value0x11 (Transparent data cluster ID).	0 - 0xFFFF	0x11
NP	Maximum RF Payload Bytes. This value returns the maximum number of RF payload bytes that can be sent in a unicast transmission. If APS encryption is used (API transmit option bit enabled), the maximum payload size is reduced by 9 bytes. If source routing is used (AR < 0xFF), the maximum payload size is reduced further. Note: NP returns a hexadecimal value. (e.g. if NP returns 0x54, this is equivalent to 84 bytes)	0 - 0xFFFF	[read-only]
DD	Device Type Identifier. Stores a device type value. This value can be used to differentiate different XBee-based devices. Digi reserves the range 0 - 0xFFFFFF. For the XBee ZB SMT module, the device type is 0xA0000.	0 - 0xFFFFFFFF	0xA0000

Networking

Networking Commands

AT Command	Name and Description	Parameter Range	Default
CH	Operating Channel. Read the channel number used for transmitting and receiving between RF modules. Uses 802.15.4 channel numbers. A value of 0 means the device has not joined a PAN and is not operating on any channel.	XBee 0, 0x0B - 0x1A XBee-PRO 0, 0x0B - 0x19 (Channels 11-25)	[read-only]
CE	Coordinator Enable. Set/read whether module is a coordinator.	0 - Not a coordinator 1 - Coordinator (SM must be 0 in order to set CE to 1.)	0
ID	Extended PAN ID. Set/read the 64-bit extended PAN ID. If set to 0, the coordinator will select a random extended PAN ID, and the router / end device will join any extended PAN ID. Changes to ID should be written to non-volatile memory using the WR command to preserve the ID setting if a power cycle occurs.	0 - 0xFFFFFFFFFFFFFFFF	0
OP	Operating Extended PAN ID. Read the 64-bit extended PAN ID. The OP value reflects the operating extended PAN ID that the module is running on. If ID > 0, OP will equal ID.	0x01 - 0xFFFFFFFFFFFFFFFF	[read-only]
NH	Maximum Unicast Hops. Set / read the maximum hops limit. This limit sets the maximum broadcast hops value (BH) and determines the unicast timeout. The timeout is computed as (50 * NH) + 100 ms. The default unicast timeout of 1.6 seconds (NH=0x1E) is enough time for data and the acknowledgment to traverse about 8 hops.	0 - 0xFF	0x1E
BH	Broadcast Hops. Set/Read the maximum number of hops for each broadcast data transmission. Setting this to 0 will use the maximum number of hops.	0 - 0x1E	0
OI	Operating 16-bit PAN ID. Read the 16-bit PAN ID. The OI value reflects the actual 16-bit PAN ID the module is running on.	0 - 0xFFFF	[read-only]
NT	Node Discovery Timeout. Set/Read the node discovery timeout. When the network discovery (ND) command is issued, the NT value is included in the transmission to provide all remote devices with a response timeout. Remote devices wait a random time, less than NT, before sending their response.	0x20 - 0xFF [x 100 msec]	0x3C (60d)
NO	Network Discovery options. Set/Read the options value for the network discovery command. The options bitfield value can change the behavior of the ND (network discovery) command and/or change what optional values are returned in any received ND responses or API node identification frames. Options include: 0x01 = Append DD value (to ND responses or API node identification frames) 002 = Local device sends ND response frame when ND is issued.	0 - 0x03 [bitfield]	0
SC	Scan Channels. Set/Read the list of channels to scan. *Coordinator* - Bit field list of channels to choose from prior to starting network. *Router/End Device* - Bit field list of channels that will be scanned to find a Coordinator/ Router to join. Changes to SC should be written using WR command to preserve the SC setting if a power cycle occurs. Bit (Channel): 0 (0x0B) 4 (0x0F) 8 (0x13) 12 (0x17) 1 (0x0C) 5 (0x10) 9 (0x14) 13 (0x18) 2 (0x0D) 6 (0x11) 10 (0x15) 14 (0x19) 3 (0x0E) 7 (0x12) 11 (0x16) 15 (0x1A)	XBee 1 - 0xFFFF [bitfield] XBee-PRO 1-0x7FFF (bit 15 is not allowed)	7FFF
SD	Scan Duration. Set/Read the scan duration exponent. Changes to SD should be written using WR command. *Coordinator* - Duration of the Active and Energy Scans (on each channel) that are used to determine an acceptable channel and Pan ID for the Coordinator to startup on. *Router / End Device* - Duration of Active Scan (on each channel) used to locate an available Coordinator / Router to join during Association. Scan Time is measured as:(# Channels to Scan) * (2 ^ SD) * 15.36ms - The number of channels to scan is determined by the SC parameter. The XBee can scan up to 16 channels (SC = 0xFFFF). Sample Scan Duration times (13 channel scan): If SD = 0, time = 0.200 sec SD = 2, time = 0.799 sec SD = 4, time = 3.190 sec SD = 6, time = 12.780 sec Note: SD influences the time the MAC listens for beacons or runs an energy scan on a given channel. The SD time is not a good estimate of the router/end device joining time requirements. ZigBee joining adds additional overhead including beacon processing on each channel, sending a join request, etc. that extend the actual joining time.	0 - 7 [exponent]	3
ZS	ZigBee Stack Profile. Set / read the ZigBee stack profile value. This must be set the same on all devices that should join the same network.	0 - 2	0

Networking Commands

AT Command	Name and Description	Parameter Range	Default
NJ	Node Join Time. Set/Read the time that a Coordinator/Router allows nodes to join. This value can be changed at run time without requiring a Coordinator or Router to restart. The time starts once the Coordinator or Router has started. The timer is reset on power-cycle or when NJ changes. For an end device to enable rejoining, NJ should be set less than 0xFF on the device that will join. If NJ < 0xFF, the device assumes the network is not allowing joining and first tries to join a network using rejoining. If multiple rejoining attempts fail, or if NJ=0xFF, the device will attempt to join using association.	0 - 0xFF [x 1 sec]	0xFF (always allows joining)
JV	Channel Verification. Set/Read the channel verification parameter. If JV=1, a router will verify the coordinator is on its operating channel when joining or coming up from a power cycle. If a coordinator is not detected, the router will leave its current channel and attempt to join a new PAN. If JV=0, the router will continue operating on its current channel even if a coordinator is not detected.	0 - Channel verification disabled 1 - Channel verification enabled	0
NW	Network Watchdog Timeout. Set/read the network watchdog timeout value. If NW is set > 0, the router will monitor communication from the coordinator (or data collector) and leave the network if it cannot communicate with the coordinator for 3 NW periods. The timer is reset each time data is received from or sent to a coordinator, or if a many-to-one broadcast is received.	0 - 0x64FF [x 1 minute] (up to over 17 days)	0 (disabled)
JN	Join Notification. Set / read the join notification setting. If enabled, the module will transmit a broadcast node identification packet on power up and when joining. This action blinks the Associate LED rapidly on all devices that receive the transmission, and sends an API frame out the serial port of API devices. This feature should be disabled for large networks to prevent excessive broadcasts.	0 - 1	0
AR	Aggregate Routing Notification. Set/read time between consecutive aggregate route broadcast messages. If used, AR should be set on only one device to enable many-to-one routing to the device. Setting AR to 0 only sends one broadcast	0 - 0xFF	0xFF

Security

Security Commands

AT Command	Name and Description	Parameter Range	Default
EE	Encryption Enable. Set/Read the encryption enable setting.	0 - Encryption disabled 1 - Encryption enabled	0
EO	Encryption Options. Configure options for encryption. Unused option bits should be set to 0. Options include: 0x01 - Send the security key unsecured over-the-air during joins 0x02 - Use trust center (coordinator only	0 - 0xFF	
NK	Network Encryption Key. Set the 128-bit AES network encryption key. This command is write-only; NK cannot be read. If set to 0 (default), the module will select a random network key.	128-bit value	0
KY	Link Key. Set the 128-bit AES link key. This command is write only; KY cannot be read. Setting KY to 0 will cause the coordinator to transmit the network key in the clear to joining devices, and will cause joining devices to acquire the network key in the clear when joining.	128-bit value	0

RF Interfacing

RF Interfacing Commands

AT Command	Name and Description	Parameter Range	Default
PL	Power Level. Select/Read the power level at which the RF module transmits conducted power. For XBee-PRO (S2B) Power Level 4 is calibrated and the other power levels are approximate. Calibration occurs every 15 seconds based on radio characteristics determined at manufacturing time, the ambient temperature, and how far off the voltage is from the typical 3.3 V. If the input voltage is too high, the module will reset. For the regular XBee, when operating on channel 26, no PL/PM selection will allow greater than +3 dBm output.	XBee (boost mode disabled) 0 = -5 dBm 1 = -1 dBm 2 = +1 dBm 3 = +3 dBm 4 = +5 dBm XBee-PRO (Boost mode enabled) 4 =+18 dBM 3 = +16 dBm (approx.) 2 = +14 dBm (approx.) 1 = +12 dBm (approx.) 0 = 0 dBm (approx.)	4
PM	Power Mode (XBee only). Set/read the power mode of the device. Enabling boost mode will improve the receive sensitivity by 2dB and increase the transmit power by 3dB Note:This command is disabled on the XBee-PRO. It is forced on by the software to provide the extra sensitivity. Boost mode imposes a slight increase in current draw. See section 1.2 for details.	0-1, 0= -Boost mode disabled, 1= Boost mode enabled.	1
DB	Received Signal Strength. This command reports the received signal strength of the last received RF data packet or APS acknowledgment. The DB command only indicates the signal strength of the last hop. It does not provide an accurate quality measurement for a multihop link. DB can be set to 0 to clear it. The DB command value is measured in -dBm. For example if DB returns 0x50, then the RSSI of the last packet received was -80dBm.	0 - 0xFF Observed range for XBee-PRO: 0x1A - 0x58 For XBee: 0x 1A - 0x5C	
PP	Peak Power. Read the dBm output when maximum power is selected (PL4).	0x0-0x12	[read only]

Serial Interfacing (I/O)

Serial Interfacing Commands

AT Command	Name and Description	Parameter Range	Default
AP	API Enable. Enable API Mode. This command is ignored when using SPI. API mode 1 is always used.	0 = API-disabled (operate in transparent mode) 1 = API-enabled 2 = API-enabled (w/escaped control characters)	1
AO	API Options. Configure options for API. Current options select the type of receive API frame to send out the Uart for received RF data packets.	0 - Default receive API indicators enabled 1 - Explicit Rx data indicator API frame enabled (0x91) 3 - enable ZDO passthrough of ZDO requests to the serial port which are not supported by the stack, as well as Simple_Desc_req, Active_EP_req, and Match_Desc_req.	0
BD	Interface Data Rate. Set/Read the serial interface data rate for communication between the module serial port and host. Any value above 0x0A will be interpreted as an actual baud rate. Standard baud rates are supported. Non-standard baud rates are permitted but their performance is not guaranteed.	0 - 0x0A (standard baud rates) 0 = 1200 bps 1 = 2400 2 = 4800 3 = 9600 4 = 19200 5 = 38400 6 = 57600 7 = 115200 8 = 230400 9 = 460800 A = 921600	3
NB	Serial Parity. Set/Read the serial parity setting on the UART.	0 = No parity 1 = Even parity 2 = Odd parity 3 = Mark parity	0
SB	Stop Bits. Set/read the number of stop bits for the UART. (Two stop bits are not supported if mark parity is enabled.)	0 = 1 stop bit 1 = 2 stop bits	0
RO	Packetization Timeout. Set/Read number of character times of inter-character silence required before packetization. Set (RO=0) to transmit characters as they arrive instead of buffering them into one RF packet The RO command is only supported when operating in transparent mode.	0 - 0xFF [x character times]	3
D7	DIO7 Configuration. Select/Read options for the DIO7 line of the RF module.	0 = Unmonitored digital input 1 = CTS Flow Control 3 = Digital input 4 = Digital output, low 5 = Digital output, high 6 = RS-485 transmit enable (low enable) 7 = RS-485 transmit enable (high enable)	1
D6	DIO6 Configuration. Configure options for the DIO6 line of the RF module.	0 = Unmonitored digital input 1 = RTS flow control 3 = Digital input 4 = Digital output, low 5 = Digital output, high	0

I/O Commands

I/O Commands

AT Command	Name and Description	Parameter Range	Default
IR	I/O Sample Rate. Set/Read the I/O sample rate to enable periodic sampling. For periodic sampling to be enabled, IR must be set to a non-zero value, and at least one module pin must have analog or digital I/O functionality enabled (see D0-D9, P0-P4 commands). The sample rate is measured in milliseconds.	0, 0x32:0xFFFF (ms)	0
IC	I/O Digital Change Detection. Set/Read the digital I/O pins to monitor for changes in the I/O state. IC works with the individual pin configuration commands (D0-D9, P0-P4). If a pin is enabled as a digital input/output, the IC command can be used to force an immediate I/O sample transmission when the DIO state changes. IC is a bitmask that can be used to enable or disable edge detection on individual channels. Unused bits should be set to 0. Bit (IO pin): 0 (DIO0)4 (DIO4)8 (DIO8) 1 (DIO1) 5 (DIO5) 9 (DIO9) 2 (DIO2) 6 (DIO6) 10 (DIO10) 3 (DIO3) 7 (DIO7) 11 (DIO11)	: 0 - 0xFFFF	0
P0	PWM0 Configuration. Select/Read function for PWM0.	0 = Unmonitored digital input 1 = RSSI PWM 3 - Digital input, monitored 4 - Digital output, default low 5 - Digital output, default high	1
P1	PWM1 / DIO11 Configuration. Configure options for the DIO11 line of the RF module.	0 - Unmonitored digital input 1 - Output 50% duty cycle clock at 32.787 kHz 3- Digital input, monitored 4- Digital output, default low 5- Digital output, default high	0
P2	DIO12 Configuration. Configure options for the DIO12 line of the RF module.	0 - Unmonitored digital input 3- Digital input, monitored 4- Digital output, default low 5- Digital output, default high	0
P3	DIO13 / DOUT Configuration. Set/Read function for DIO13. Configure options for the DIO13 line of the RF module.	0 – Unmonitored digital input 1 – Data out for UART 3 – Monitored digital input 4 – Digital output low 5 – Digital output high	1
P4	DIO14 / DIN. Set/read function for DIO14.	0 – Unmonitored digital input 1 – Data in for UART 3 – Digital input 4 – Digital output low 5 – Digital output high	1
P5	DIO15 / SPI_MISO. Set/read function for DIO15.	0 – Unmonitored digital input 1 – Output from SPI port	1
P6	DIO16 / SPI_MOSI. Set/read function for DIO16.	0 – Unmonitored digital input 1 – Input to SPI port	1
P7	DIO17 / SPI_$\overline{\text{SSEL}}$. Set/read function for DIO17.	0 – Unmonitored digital input 1 – Input to to select the SPI port	1
P8	DIO18 / SPI_SClk. Set/read function for DIO18.	0 – Unmonitored digital input 1 – SPI clock input	1

I/O Commands

AT Command	Name and Description	Parameter Range	Default
P9	DIO19 / SPI_Attn / PTI_DATA. Set/read function for DIO19.	0 – Unmonitored digital input 1 - SPI data available indicator 6 – Packet trace interface data output. Must be set along with D1=6 to output traces for OTA sniffing.	1
D0	AD0/DIO0 Configuration. Select/Read function for AD0/DIO0.	0 - Unmonitored digital input 1 - Commissioning button enabled 2 - Analog input, single ended 3 - Digital input 4 - Digital output, low 5 - Digital output, high	1
D1	AD1/DIO1 / PTI_En Configuration. Select/Read function for AD1/DIO1.	0 – Unmonitored digital input 2 - Analog input, single ended 3 – Digital input 4 – Digital output, low 5 – Digital output, high 6 - Packet trace interface enable. Must be set along with P9=6 to output traces for OTA sniffing.	0
D2	AD2/DIO2 Configuration. Select/Read function for AD2/DIO2.	0 – Unmonitored digital input 2 - Analog input, single ended 3 – Digital input 4 – Digital output, low 5 – Digital output, high	0
D3	AD3/DIO3 Configuration. Select/Read function for AD3/DIO3.	0 – Unmonitored digital input 2 - Analog input, single ended 3 – Digital input 4 – Digital output, low 5 – Digital output, high	0
D4	DIO4 Configuration. Select/Read function for DIO4.	0 – Unmonitored digital input 3 – Digital input 4 – Digital output, low 5 – Digital output, high	0
D5	DIO5 / Associate Configuration. Configure options for the DIO5 line of the RF module.	0 - Unmonitored digital input 1 - Associated indication LED 3 - Digital input 4 - Digital output, default low 5 - Digital output, default high	1
D8	DIO8 / DTR / Slp_Rq. Set/Read function for DIO8.	0 – Unmonitored digital input 1 – Input to sleep and wake module 3 – Digital input 4 – Digital output, low 5 – Digital output, high	
LT	Assoc LED Blink Time. Set/Read the Associate LED blink time. If the Associate LED functionality is enabled (D5 command), this value determines the on and off blink times for the LED when the module has joined a network. If LT=0, the default blink rate will be used (500ms coordinator, 250ms router/end device). For all other LT values, LT is measured in 10ms.	0, 0x0A - 0xFF (100 - 2550 ms)	0

I/O Commands

AT Command	Name and Description	Parameter Range	Default
PR	Pull-up/down Resistor. Set/read the bit field that configures the internal pull-up/down resistor status for the I/O lines. "1" specifies the pull-up/down resistor is enabled. "0" specifies no internal resistors are used. The input will be floating. Bits:" 0 - DIO4 (Pin 24) 1 - AD3 / DIO3 (Pin 30) 2 - AD2 / DIO2 (Pin 31) 3 - AD1 / DIO1 (Pin 32) 4 - AD0 / DIO0 (Pin 33) 5 - RTS / DIO6 (Pin 29) 6 - DTR / Sleep Request / DIO8 (Pin 10) 7 - DIN / Config (Pin 4) 8 - Associate / DIO5 (Pin 28) 9 - On/Sleep / DIO9 (Pin 26) 10 - DIO12 (Pin 5) 11 - PWM0 / RSSI / DIO10 (Pin 7) 12 - PWM1 / DIO11 (Pin 8) 13 - CTS / DIO7 (Pin 25) 14 - DOUT / DIO13 (Pin 3)	0 - 0x7FFF	0x1FFF
PD	Pull-up / down direction. Set/read an internal pull-up or pull-down resistor for the corresponding bits in the PR command. If the bit is set, an internal pull-up resistor is used. If it is clear, an internal pull-down resistor is used. See the PR command for the bit order.	0 - 0x7FFF	0x1FBF
RP	RSSI PWM Timer. Time the RSSI signal will be output on the PWM after the last RF data reception or APS acknowledgment.. When RP = 0xFF, output will always be on.	0 - 0xFF [x 100 ms]	0x28 (40d)
DO	Device Options. Bit0 - Reserved. Bit1 - Reserved for Smart Energy devices. Bit2 - 0/1 = First or Best Join. First join means the device will join the network through the first acceptable Beacon response it receives. Best join means the device will join the network through the strongest Beacon response it receives after searching all search mask channels. Bit3 - Disable NULL Transport Key (Coordinator Only). Bit4 - Disable Tx Packet Extended Timeout. Bit5 - Disable ACK for End Device I/O Sampling. Bit6 - Enable High Ram Concentrator.	0x00-0xFF	0x00
%V	Supply Voltage. Reads the voltage on the Vcc pin in mV.	-0x-0xFFFF [read only]	-
V+	Voltage Supply Monitoring. The voltage supply threshold is set with the V+ command. If the measured supply voltage falls below or equal to this threshold, the supply voltage will be included in the IO sample set. V+ is set to 0 by default (do not include the supply voltage).The units of this command are mV. For example, to include a measurement of the supply voltage when it exceeds 3.3 V, set V+ to 3300 = 0xCE4.	0-0xFFFF	0
TP	Reads the module temperature in Degrees Celsius. Accuracy +/- 7 degrees. 1° C = 0x0001 and -1° C = 0xFFFF. Command is only available on PRO module.	0x0-0xFFFF	-

Diagnostics

Diagnostics Commands

AT Command	Name and Description	Parameter Range	Default
VR	Firmware Version. Read firmware version of the module as a 4-digit hex number.	0 - 0xFFFF [read-only]	Factory-set
VL	Version Long. Shows detailed version information, module type, a time stamp for the build, Ember stack version, and bootloader version.	N/A	N/A
HV	Hardware Version. Read the hardware version of the module.version of the module. This command can be used to distinguish among different hardware platforms. The upper byte returns a value that is unique to each module type. The lower byte indicates the hardware revision. The regular XBee returns a value of 0x22xx for this command. the XBee-PRO returns a value of 0x21xx.	0 - 0xFFFF [read-only]	Factory-set
AI	Association Indication. Read information regarding last node join request: 0x00 - Successfully formed or joined a network. (Coordinators form a network, routers and end devices join a network.) 0x21 - Scan found no PANs 0x22 - Scan found no valid PANs based on current SC and ID settings 0x23 - Valid Coordinator or Routers found, but they are not allowing joining (NJ expired) 0x24 - No joinable beacons were found 0x25 - Unexpected state, node should not be attempting to join at this time 0x27 - Node Joining attempt failed (typically due to incompatible security settings) 0x2A - Coordinator Start attempt failed' 0x2B - Checking for an existing coordinator 0x2C - Attempt to leave the network failed 0xAB - Attempted to join a device that did not respond. 0xAC - Secure join error - network security key received unsecured 0xAD - Secure join error - network security key not received 0xAF - Secure join error - joining device does not have the right preconfigured link key 0xFF - Scanning for a ZigBee network (routers and end devices) Note: New non-zero AI values may be added in later firmware versions. Applications should read AI until it returns 0x00, indicating a successful startup (coordinator) or join (routers and end devices)	0 - 0xFF [read-only]	--

AT Command Options

AT Command Options Commands

AT Command	Name and Description	Parameter Range	Default
CT	Command Mode Timeout. Set/Read the period of inactivity (no valid commands received) after which the RF module automatically exits AT Command Mode and returns to Idle Mode.	2 - 0x028F [x 100 ms]	0x64 (100d)
CN	Exit Command Mode. Explicitly exit the module from AT Command Mode.	--	--
GT	Guard Times. Set required period of silence before and after the Command Sequence Characters of the AT Command Mode Sequence (GT + CC + GT). The period of silence is used to prevent inadvertent entrance into AT Command Mode.	1 - 0x0CE4 [x 1 ms] (max of 3.3 decimal sec)	0x3E8 (1000d)
CC	Command Sequence Character. Set/Read the ASCII character value to be used between Guard Times of the AT Command Mode Sequence (GT + CC + GT). The AT Command Mode Sequence enters the RF module into AT Command Mode.	0 - 0xFF	0x2B ('+' ASCII)

Sleep Commands

Sleep Commands

AT Command	Name and Description	Parameter Range	Default
SM	Sleep Mode Sets the sleep mode on the RF module. When SM>0, the module operates as an end device. However, CE must be 0 before SM can be set to a value greater than 0 to turn the module into an end device. Changing a device from a router to an end device (or vice versa) forces the device to leave the network and attempt to join as the new device type when changes are applied.	0-Sleep disabled (router) 1-Pin sleep enabled 4-Cyclic sleep enabled 5 - Cyclic sleep, pin wake	0 - Router 4 - End Device
SN	Number of Sleep Periods. Sets the number of sleep periods to not assert the On/Sleep pin on wakeup if no RF data is waiting for the end device. This command allows a host application to sleep for an extended time if no RF data is present	1 - 0xFFFF	1
SP	Sleep Period. This value determines how long the end device will sleep at a time, up to 28 seconds. (The sleep time can effectively be extended past 28 seconds using the SN command.) On the parent, this value determines how long the parent will buffer a message for the sleeping end device. It should be set at least equal to the longest SP time of any child end device.	0x20 - 0xAF0 x 10ms (Quarter second resolution)	0x20
ST	Time Before Sleep Sets the time before sleep timer on an end device.The timer is reset each time serial or RF data is received. Once the timer expires, an end device may enter low power operation. Applicable for cyclic sleep end devices only.	1 - 0xFFFE (x 1ms)	0x1388 (5 seconds)
SO Command	Sleep Options. Configure options for sleep. Unused option bits should be set to 0. Sleep options include: 0x02 - Always wake for ST time 0x04 - Sleep entire SN * SP time Sleep options should not be used for most applications. See chapter 6 for more information.	0 - 0xFF	0
WH	Wake Host. Set/Read the wake host timer value. If the wake host timer is set to a non-zero value, this timer specifies a time (in millisecond units) that the device should allow after waking from sleep before sending data out the serial port or transmitting an I/O sample. If serial characters are received, the WH timer is stopped immediately.	0 - 0xFFFF (x 1ms)	
SI	Sleep Immediately. See Execution Commands table below..		
PO	Polling Rate. Set/Read the end device poll rate. Setting this to 0 (default) enables polling at 100 ms (default rate), advancing in 10 msec increments. Adaptive polling may allow the end device to poll more rapidly for a short time when receiving RF data.	0 - 0x3E8	0x00 (100 msec)

Execution Commands

Where most AT commands set or query register values, execution commands cause an action to be executed on the module. Execution commands are executed immediately and do not require changes to be applied.

AT Command	Name and Description	Parameter Range	Default
AC	Apply Changes. Applies changes to all command registers causing queued command register values to be applied. For example, changing the serial interface rate with the BD command will not change the UART interface rate until changes are applied with the AC command. The CN command and 0x08 API command frame also apply changes.	-	
AS	Active Scan. Scans the neighborhood for beacon responses. Response frames are structured as: AS_type – unsigned byte = 2 - ZB firmware uses a different format than WiFi XBee, which is type 1 Channel – unsigned byte PAN – unsigned word in big endian format Extended PAN – eight unsigned bytes in bit endian format Allow Join – unsigned byte – 1 indicates join is enabled, 0 that it is disabled Stack Profile – unsigned byte LQI – unsigned byte, higher values are better RSSI – signed byte, lower values are better		
WR	Write. Write parameter values to non-volatile memory so that parameter modifications persist through subsequent resets. Note: Once WR is issued, no additional characters should be sent to the module until after the "OK\r" response is received. The WR command should be used sparingly. The EM357 supports a limited number of write cycles.	--	--
RE	Restore Defaults. Restore module parameters to factory defaults.	--	--
FR	Software Reset. Reset module. Responds immediately with an OK status, and then performs a software reset about 2 seconds later.	--	--

AT Command	Name and Description	Parameter Range	Default
NR	Network Reset. Reset network layer parameters on one or more modules within a PAN. Responds immediately with an "OK" then causes a network restart. All network configuration and routing information is consequently lost. *If NR = 0*: Resets network layer parameters on the node issuing the command. *If NR = 1*: Sends broadcast transmission to reset network layer parameters on all nodes in the PAN.	0 - 1	--
SI	Sleep Immediately. Cause a cyclic sleep module to sleep immediately rather than wait for the ST timer to expire.	-	-
CB	Commissioning Pushbutton. This command can be used to simulate commissioning button presses in software. The parameter value should be set to the number of button presses to be simulated. For example, sending the ATCB1 command will execute the action associated with 1 commissioning button press.		
&X	Clear Binding and Group Tables. This command resets the binding and group tables.		
ND	Node Discover. Discovers and reports all RF modules found. The following information is reported for each module discovered. MY<CR> SH<CR> SL<CR> NI<CR> (Variable length) PARENT_NETWORK ADDRESS (2 Bytes)<CR> DEVICE_TYPE<CR> (1 Byte: 0=Coord, 1=Router, 2=End Device) STATUS<CR> (1 Byte: Reserved) PROFILE_ID<CR> (2 Bytes) MANUFACTURER_ID<CR> (2 Bytes) <CR> After (NT * 100) milliseconds, the command ends by returning a <CR>. ND also accepts a Node Identifier (NI) as a parameter (optional). In this case, the first module with a matching NI identifier to respond will be returned. If no module matches, then "ERROR" will be returned. If ND is sent through the API, each response is returned as a separate AT_CMD_Response packet. The data consists of the above listed bytes without the carriage return delimiters. The NI string will end in a "0x00" null character. The radius of the ND command is set by the BH command. Refer to the description of the NO command for options which affect the behavior of the ND command.	optional 20-Byte NI or MY value	--
DN	Destination Node. Resolves an NI (Node Identifier) string to a physical address (case-sensitive). The following events occur after the destination node is discovered: <AT Firmware> 1. DL & DH are set to the extended (64-bit) address of the module with the matching NI (Node Identifier) string. 2. OK (or ERROR)\r is returned. 3. Command Mode is exited to allow immediate communication <API Firmware> 1. The 16-bit network and 64-bit extended addresses are returned in an API Command Response frame. If there is no response from a module within (NT * 100) milliseconds or a parameter is not specified (left blank), the command is terminated and an "ERROR" message is returned. In the case of an ERROR, Command Mode is not exited. The radius of the DN command is set by the BH command.	up to 20-Byte printable ASCII string	--
IS	Force Sample Forces a read of all enabled digital and analog input lines.	--	--

11. Module Support

This chapter provides customization information for the XBee. In addition to providing an extremely flexible and powerful API, XBee modules are a robust development platform that have passed FCC and ETSI testing. Developers can customize default parameters, or even write or load custom firmware for Ember's EM357 chip.

X-CTU Configuration Tool

Digi provides a Windows X-CTU configuration tool for configuring module parameters and updating firmware. The XCTU has the capability to do the following:

- Discover all XBee devices in the network
- Update firmware on a local module (requires USB or serial connection)
- Read or write module configuration parameters on a local or remote device
- Save and load configuration profiles containing customized settings.

Contact Digi support for more information about the X-CTU.

Customizing XBee ZB Firmware

Once module parameters are tested in an application and finalized, Digi can manufacture modules with specific, customer-defined configurations for a nominal fee. These custom configurations can lock in a firmware version or set command values when the modules are manufactured, eliminating the need for customers to adjust module parameters on arrival. Alternatively, Digi can program custom firmware, including Ember's EZSP UART image, into the modules during manufacturing. Contact Digi to create a custom configuration.

Design Considerations for Digi Drop-In Networking

XBee RF modules contain a variety of features that allow for interoperability with Digi's full line of Drop-in Networking products. Interoperability with other "DIN" products can offer these advantages:

- Add IP-connectivity to your network via Cellular, Ethernet or WiFi with a ConnectPort X Gateway.
- Extend the range of your network with the XBee Wall Router.
- Make deployment easy by enabling the Commissioning Pushbutton (pin 20) and AssociateLED (pin 15) to operate with the Network Commissioning Tool software.
- Interface with standard RS-232, USB, Analog & Digital I/O, RS-485, and other industrial devices using XBee Adapters.
- Monitor and manage your network securely from remote locations with Device Cloud by Etherios™.
- We encourage you to contact our technical representatives for consideration, implementation, or design review of your product for interoperability with Digi's Drop-in Networking solutions.

XBee Bootloader

XBee modules use a modified version of Ember's bootloader. This bootloader version supports a custom entry mechanism that uses module pins DIN (pin 4), $\overline{\text{DTR}}$ / SLEEP_RQ (pin 10), and $\overline{\text{RTS}}$ (pin 29). To invoke the bootloader, do the following:

1. Set $\overline{\text{DTR}}$ / SLEEP_RQ low (TTL 0V) and RTS high.
2. Send a serial break to the DIN pin and power cycle or reset the module.
3. When the module powers up, $\overline{\text{DTR}}$ / SLEEP_RQ and DIN should be low (TTL 0V) and RTS should be high.
4. Terminate the serial break and send a carriage return at 115200bps to the module.
5. If successful, the module will send the Ember bootloader menu out the DOUT pin at 115200bps.
6. Commands can be sent to the bootloader at 115200bps.

Note: Hardware flow control should be disabled when entering and communicating with the Ember 357 bootloader.

Programming XBee Modules

Firmware on the XBee modules can be updated serially.

Serial Firmware Updates

Serial firmware updates make use of the XBee custom bootloader which ships in all units. This modified bootloader is based on Ember's standalone bootloader, but with a modified entry mechanism. The modified entry mechanism uses module pins 4, 10, and 29 (DIN, DTR, and $\overline{\text{RTS}}$ respectively).

The X-CTU program can update firmware serially on the XBee. Contact Digi support for details.

If an application requires custom firmware to update the XBee firmware serially, the following steps are required.

Invoke XBee Bootloader

See the "XBee Bootloader" section above for steps to invoke the bootloader using RS-232 signals. The bootloader may also be invoked by issuing a command via X-CTU. Then the application makes an explicit call to the bootloader, which does not return.

If there is no valid application, the bootloader will always run.

Send Firmware Image

After invoking the bootloader, the Ember bootloader will send the bootloader menu characters out the serial port, which may be the UART at 115200 bps or the SPI, where the attached SPI master provides the clock rate. The application should do the following to upload a firmware image.

1. Look for the bootloader prompt "BL >" to ensure the bootloader is active
2. Send an ASCII "1" character to initiate a firmware update
3. After sending a "1", the EM357 waits for an XModem CRC upload of an .ebl image over the serial line at 115200 bps. The .ebl file must be sent to the EM357 in order.

If the firmware image is successfully loaded, the bootloader will output a “complete” string. Then the newly loaded firmware can be invoked by sending a ‘2’ to the module.

If the firmware image is not successfully loaded, the bootloader will output an “aborted” string. Then it will return to the main bootloader menu. Some causes for failure are:

- Over 1 minute passes after the command to send the firmware image and the first block of the image has not yet been sent.
- A power cycle or reset event occurs during the firmware load.
- A file error or a flash error occurs during the firmware load.

Writing Custom Firmware

The XBee module can be used as a hardware development platform for the EM357. Custom firmware images can be developed around the EmberZNet 4.2.xx mesh stacks (for the EM357) and uploaded to the XBee.

Warning: If programming firmware through the JTAG interface, please be aware that doing so can potentially erase the XBee bootloader. If this occurs, serial firmware updates will not work.

Regulatory Compliance

XBee modules are FCC and ETSI certified for operation on all 16 channels. The EM357 output power can be configured up to 8 dBm with boost mode enabled.

XBee-PRO modules are certified for operation on 15 of the 16 band channels (channels 11 - 25). The scan channels mask of XBee-PRO devices must be set in the application to disable the upper channel (e.g. 0x03FFF800). The XBee-PRO contains a power compensation method to adjust the output power near 18 dBm. For best results, the EM357 should be configured with an output power level of -4 dBm with Boost mode is enabled. The end product is responsible to adhere to these requirements.

Enabling GPIO 1 and 2

Most of the remaining sections in this chapter describe how to configure GPIOs to function correctly in custom applications that run on the XBee modules. In order for GPIO pins to be configurable, the application must set the GPIO_PxCFG registers to enable the appropriate GPIO. The following table lists values for configuring the GPIO pins. Other functionality is affected by these settings. See the EM357 datasheet from Ember for a complete listing of functionality.

GPIO Mode	GPIO_PxCFGH/L	Description
Analog	0x0	Analog input or output. When in analog mode, the digital input (GPIO_PxIN) always reads 1.
Input (floating)	0x4	Digital input without an internal pull-up or pull-down. Output is disabled.
Input (pull-up or pull-down)	0x8	Digital input with an internal pull-up or pull-down. A set bit in GPIO_PxOUT selects pull-up and a cleared bit selects pull-down. Output is disabled
Output (push-pull)	0x1	Push-pull output. GPIO_PxOUT controls the output.
Output (open-drain)	0x5	Open-drain output. GPIO_PxOUT controls the output. If a pull-up is required, it must be external.
Alternate Output (push-pull)	0x9	Push-pull output. An on-board peripheral controls the output.
Alternate Output (open-drain)	0xD	Open-drain output. An on-board peripheral controls the output. If a pull-up is required, it must be external.

For more information on configuring and setting GPIOs, consult the EM357 specification.

Detecting XBee vs. XBee-PRO

For some applications, it may be necessary to determine if the code is running on an XBee or an XBee-PRO device. The PC7 pin on the EM357 is used to identify the module type (see Chapter 1). PC7 is connected to ground on the XBee module. The following code could be used to determine if a module is an XBee or XBee-PRO:

```
GPIO_PCSET = 0x80; // Enable pullup resistor
GPIO_PCCFGH &= 0x0fff; // Clear PC7 config
GPIO_PCCFGH |= 0x8000;// Set PC7 as input with pullup/pulldown
if (GPIO_PCIN & 0x80) {
   ModuleIsXBeePro = true;
} else {
   ModuleIsXBeePro = false;
}
```

Special Instructions For Using the JTAG Interface

There are four JTAG programming pins on the XBee through which firmware can be loaded onto the EM357 processor. Three of these pins are also connected to a second pin on the XBee and are used for separate functions. The following table indicates the JTAG signal name, the primary connection pin on the XBee, the secondary connection pin, and the secondary signal name.

It is important that the secondary pins specifically are not loaded with circuitry that might interfere with JTAG programming (for example, an LED tied directly to the ASSOCIATE / DIO5 line). Any loading circuitry should be buffered to avoid conflicts (for example, connecting ASSOCIATE / DIO5 to the gate of a MOSFET which drives the LED).

JTAG pin name	Primary XBee pin	Secondary XBee pin	Secondary pin name
JTCK	18	N/A	N/A
JTDO	19	26	ON / $\overline{\text{SLEEP}}$ / DIO9
JTDI	20	28	ASSOCIATE / DIO5
JTMS	21	5	DIO12

Appendix A: Agency Certifications

United States FCC

The XBee RF Module complies with Part 15 of the FCC rules and regulations. Compliance with the labeling requirements, FCC notices and antenna usage guidelines is required.

To fulfill FCC Certification, the OEM must comply with the following regulations:

1.The system integrator must ensure that the text on the external label provided with this device is placed on the outside of the final product.

2.XBee RF Module may only be used with antennas that have been tested and approved for use with this module [refer to the antenna tables in this section].

OEM Labeling Requirements

WARNING: The Original Equipment Manufacturer (OEM) must ensure that FCC labeling requirements are met. This includes a clearly visible label on the outside of the final product enclosure that displays the contents shown in the figure below.

Required FCC Label for OEM products containing the XBee S2C RF Module

> Contains FCC ID: MCQ-XBS2C
>
> This device complies with Part 15 of the FCC Rules. Operation is subject to the following two conditions: (1.) this device may not cause harmful interference and (2.) this device must accept any interference received, including interference that may cause undesired operation.

Required FCC Label for OEM products containing the XBee-PRO S2C RF Module

> Contains FCC ID:MCQ-XBPS2C
>
> This device complies with Part 15 of the FCC Rules. Operation is subject to the following two conditions: (1.) this device may not cause harmful interference and (2.) this device must accept any interference received, including interference that may cause undesired operation.

FCC Notices

IMPORTANT: The XBee and XBee-PRO RF Module have been certified by the FCC for use with other products without any further certification (as per FCC section 2.1091). Modifications not expressly approved by Digi could void the user's authority to operate the equipment.

IMPORTANT: OEMs must test final product to comply with unintentional radiators (FCC section 15.107 & 15.109) before declaring compliance of their final product to Part 15 of the FCC Rules.

IMPORTANT: The RF module has been certified for remote and base radio applications. If the module will be used for portable applications, the device must undergo SAR testing.

This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation.

If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures: Re-orient or relocate the receiving antenna, Increase the separation between the equipment and receiver, Connect equipment and receiver to outlets on different circuits, or Consult the dealer or an experienced radio/TV technician for help.

FCC-Approved Antennas (2.4 GHz)

The XBee and XBee-PRO RF Module can be installed utilizing antennas and cables constructed with standard connectors (Type-N, SMA, TNC, etc.).

The modules are FCC approved for fixed base station and mobile applications for the channels indicated in the tables below. If the antenna is mounted at least 20cm (8 in.) from nearby persons, the application is considered a mobile application. Antennas not listed in the table must be tested to comply with FCC Section 15.203 (Unique Antenna Connectors) and Section 15.247 (Emissions).

XBee RF Modules: XBee RF Modules have been tested and approved for use with all the antennas listed in the tables below. (Cable-loss IS required when using gain antennas as shown below.)

The antennas in the tables below have been approved for use with this module. Digi does not carry all of these antenna variants. Contact Digi Sales for available antennas.

Antennas approved for use with the XBee®/XBee-PRO® RF Modules (Cable loss is not required.)

OMNI-DIRECTIONAL ANTENNAS for All Available Channels					
Part Number	Type (Description)	Gain	Application	Min Separation	Minimum Cable Loss/Power Reduction/ Attenuation Required
29000313	Integral PCB antenna	0.0 dBi	Fixed/Mobile	20 cm	N/A
A24-HASM-450	Dipole (Half-wave articulated RPSMA - 4.5")	2.1 dBi	Fixed	20 cm	N/A
A24-HABSM	Dipole (Articulated RPSMA)	2.1 dBi	Fixed	20 cm	N/A
29000095	Dipole (Half-wave articulated RPSMA - 4.5")	2.1 dBi	Fixed/Moblie	20 cm	N/A
A24-HABUF-P5I	Dipole (Half-wave articulated bulkhead mount U.FL. w/ 5" pigtail)	2.1 dBi	Fixed/Mobile	20 cm	N/A
A24-HASM-525	Dipole (Half-wave articulated RPSMA - 5.25")	2.1 dBi	Fixed	20 cm	N/A
A24-QI	Monopole (Integrated whip)	1.5 dBi	Fixed/Mobile	20 cm	N/A
A24-F2NF	Omni-directional (Fiberglass base station)	2.1 dBi	Fixed/Mobile	20 cm	N/A
A24-F3NF	Omni-directional (Fiberglass base station)	3.0 dBi	Fixed/Mobile	20 cm	N/A
A24-F5NF	Omni-directional (Fiberglass base station)	5.0 dBi	Fixed	20 cm	N/A
A24-F8NF	Omni-directional (Fiberglass base station)	8.0 dBi	Fixed	2 m	N/A
A24-F9NF	Omni-directional (Fiberglass base station)	9.5 dBi	Fixed	2 m	N/A
A24-F10NF	Omni-directional (Fiberglass base station)	10 dBi	Fixed	2 m	N/A
A24-F12NF	Omni-directional (Fiberglass base station)	12 dBi	Fixed	2 m	N/A
A24-W7NF	Omni-directional (Fiberglass base station)	7.2 dBi	Fixed	2 m	N/A
A24-M7NF	Omni-directional (Mag-mount base station)	7.2 dBi	Fixed	2 m	N/A

Antennas approved for use with the XBee® RF Module (Cable loss is not required.)

PANEL CLASS ANTENNAS for All Available Channels					
Part Number	Type (Description)	Gain	Application	Min. Separation Required	Cable-loss
A24-P8SF	Flat Panel	8.5 dBi	Fixed	2 m	N/A
A24-P8NF	Flat Panel	8.5 dBi	Fixed	3 m	N/A
A24-P13NF	Flat Panel	13.0 dBi	Fixed	4 m	N/A
A24-P14NF	Flat Panel	14.0 dBi	Fixed	5 m	N/A

OMNI-DIRECTIONAL ANTENNAS for All Available Channels					
Part Number	Type (Description)	Gain	Application	Min. Separation Required	Cable-loss
A24-F15NF	Omni-Directional (Fiberglass base station)	15.0 dBi	Fixed	2 m	N/A

Antennas approved for use with the XBee® RF Modules (Channels 11-25)

YAGI and PANEL CLASS ANTENNAS for Channels 11 - 25					
Part Number	Type (Description)	Gain	Application*	Min. Separation Required	Cable-loss
A24-Y6NF	Yagi (6-element)	8.8 dBi	Fixed	2 m	N/A
A24-Y7NF	Yagi (7-element)	9.0 dBi	Fixed	2 m	N/A
A24-Y9NF	Yagi (9-element)	10.0 dBi	Fixed	2 m	N/A
A24-Y10NF	Yagi (10-element)	11.0 dBi	Fixed	2 m	N/A
A24-Y12NF	Yagi (12-element)	12.0 dBi	Fixed	2 m	N/A
A24-Y13NF	Yagi (13-element)	12.0 dBi	Fixed	2 m	N/A
A24-Y15NF	Yagi (15-element)	12.5 dBi	Fixed	2 m	N/A
A24-Y16NF	Yagi (16-element)	13.5 dBi	Fixed	2 m	N/A
A24-Y16RM	Yagi (16-element, RPSMA connector)	13.5 dBi	Fixed	2 m	N/A
A24-Y18NF	Yagi (18-element)	15.0 dBi	Fixed	2 m	N/A
A24-P15NF	Flat Panel	15.0 dBi	Fixed	2 m	N/A
A24-P16NF	Flat Panel	16.0 dBi	Fixed	2 m	N/A
A24-P19NF	Flat Panel	19.0 dBi	Fixed	2 m	N/A

Antennas approved for use with the XBee® RF Modules (Channel 26)

YAGI and PANEL CLASS ANTENNAS for Channel 26					
Part Number	Type (Description)	Gain	Application*	Min. Separation	Minimum Cable Loss/ Power Reduction/ Attenuation Required
A24-Y6NF	Yagi (6-element)	8.8 dBi	Fixed	2 m	N/A
A24-Y7NF	Yagi (7-element)	9.0 dBi	Fixed	2 m	N/A
A24-Y9NF	Yagi (9-element)	10.0 dBi	Fixed	2 m	N/A
A24-Y10NF	Yagi (10-element)	11.0 dBi	Fixed	2 m	N/A
A24-Y12NF	Yagi (12-element)	12.0 dBi	Fixed	2 m	N/A
A24-Y13NF	Yagi (13-element)	12.0 dBi	Fixed	2 m	0.5 dB
A24-Y15NF	Yagi (15-element)	12.5 dBi	Fixed	2 m	1.0 dB
A24-Y16NF	Yagi (16-element)	13.5 dBi	Fixed	2 m	2.0 dB
A24-Y16RM	Yagi (16-element, RPSMA connector)	13.5 dBi	Fixed	2 m	7.5 dB
A24-Y18NF	Yagi (18-element)	15.0 dBi	Fixed	2 m	9.0 dB
A24-P15NF	Flat Panel	15.0 dBi	Fixed	2 m	4.5 dB
A24-P16NF	Flat Panel	16.0 dBi	Fixed	2 m	10.0 dB
A24-P19NF	Flat Panel	19.0 dBi	Fixed	2 m	13.0 dB

Antennas approved for use with the XBee-PRO® RF Module

PANEL CLASS ANTENNAS for Channels 11- 25					
Part Number	Type (Description)	Gain	Application	Min. Separation Required	Cable-loss
A24-P8SF	Flat Panel	8.5 dBi	Fixed	2 m	N/A
A24-P8NF	Flat Panel	8.5 dBi	Fixed	2 m	N/A
A24-P13NF	Flat Panel	13.0 dBi	Fixed	2 m	4.3 dB
A24-P14NF	Flat Panel	14.0 dBi	Fixed	2 m	5.3 dB
A24-P15NF	Flat Panel	15.0 dBi	Fixed	2 m	6.3 dB
A24-P16NF	Flat Panel	16.0 dBi	Fixed	2 m	7.3 dB
A24-P19NF	Flat Panel	19.0 dBi	Fixed	2 m	10.3 dB

OMNI-DIRECTIONAL ANTENNAS for Channels 11- 25					
Part Number	Type (Description)	Gain	Application	Min. Separation Required	Cable-loss
A24-F15NF	Omni-Directional (Fiberglass base station)	15.0 dBi	Fixed	2 m	1 dB

YAGI CLASS ANTENNAS for Channels 11 - 25					
Part Number	Type (Description)	Gain	Application*	Min. Separation Required	Cable-loss
A24-Y6NF	Yagi (6-element)	8.8 dBi	Fixed	2 m	N/A
A24-Y7NF	Yagi (7-element)	9.0 dBi	Fixed	2 m	N/A
A24-Y9NF	Yagi (9-element)	10.0 dBi	Fixed	2 m	N/A
A24-Y10NF	Yagi (10-element)	11.0 dBi	Fixed	2 m	0.1 dB
A24-Y12NF	Yagi (12-element)	12.0 dBi	Fixed	2 m	1.1 dB
A24-Y13NF	Yagi (13-element)	12.0 dBi	Fixed	2 m	1.1 dB
A24-Y15NF	Yagi (15-element)	12.5 dBi	Fixed	2 m	1.6 dB
A24-Y16NF	Yagi (16-element)	13.5 dBi	Fixed	2 m	2.6 dB
A24-Y16RM	Yagi (16-element, RPSMA connector)	13.5 dBi	Fixed	2 m	2.6 dB
A24-Y18NF	Yagi (18-element)	15.0 dBi	Fixed	2 m	4.1 dB

*** If using the RF module in a portable application** (for example - if the module is used in a handheld device and the antenna is less than 20cm from the human body when the device is in operation): The integrator is responsible for passing additional SAR (Specific Absorption Rate) testing based on FCC rules 2.1091 and FCC Guidelines for Human Exposure to Radio Frequency Electromagnetic Fields, OET Bulletin and Supplement C. The testing results will be submitted to the FCC for approval prior to selling the integrated unit. The required SAR testing measures emissions from the module and how they affect the person.

RF Exposure

WARNING: To satisfy FCC RF exposure requirements for mobile transmitting devices, a separation distance of 20 cm or more should be maintained between the antenna of this device and persons during device operation. To ensure compliance, operations at closer than this distance are not recommended. The antenna used for this transmitter must not be co-located in conjunction with any other antenna or transmitter.

The preceding statement must be included as a CAUTION statement in OEM product manuals in order to alert users of FCC RF Exposure compliance.

Europe (ETSI)

The XBee RF Modules (excluding the PRO) have been certified for use in several European countries. For a complete list, refer to www.digi.com

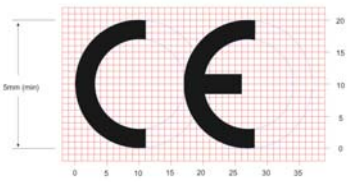
If the XBee RF Modules are incorporated into a product, the manufacturer must ensure compliance of the final product to the European harmonized EMC and low-voltage/safety standards. A Declaration of Conformity must be issued for each of these standards and kept on file as described in Annex II of the R&TTE Directive.

Furthermore, the manufacturer must maintain a copy of the XBee user manual documentation and ensure the final product does not exceed the specified power ratings, antenna specifications, and/or installation requirements as specified in the user manual. If any of these specifications are exceeded in the final product, a submission must be made to a notified body for compliance testing to all required standards.

OEM Labeling Requirements

The 'CE' marking must be affixed to a visible location on the OEM product.

CE Labeling Requirements



The CE mark shall consist of the initials "CE" taking the following form:

- If the CE marking is reduced or enlarged, the proportions given in the above graduated drawing must be respected.
- The CE marking must have a height of at least 5mm except where this is not possible on account of the nature of the apparatus.
- The CE marking must be affixed visibly, legibly, and indelibly.

Restrictions

France: Outdoor use limited to 10 mW EIRP within the band 2454-2483.5 MHz.

Norway: Norway prohibits operation near Ny-Alesund in Svalbard. More information can be found at the Norway Posts and Telecommunications site (www.npt.no).

Italy: For private use, a general authorization is required if WAS/RLANs are used outside own premises. For public use, a general authorization is required.

Russian Federation:

- Maximum mean EIRP density is 2 mW/MHz, maximum 100 mW EIRP.
- Maximum mean EIRP density is 20 mW/MHz, maximum 100 mW EIRP permitted to use SRD for outdoor applications only, for purposes of gathering telemetry information for automated monitoring and resources accounting systems or security systems.
- Maximum mean EIRP density is 10 mW/MHz, maximum 100 mW EIRP for indoor applications.

Ukraine: EIRP must be less than or equal to 100 mW with built-in antenna, with amplification factor up to 6 dBi.

Declarations of Conformity

Digi has issued Declarations of Conformity for the XBee RF Modules concerning emissions, EMC and safety. Files can be obtained by contacting Digi Support.

Important Note:

Digi does not list the entire set of standards that must be met for each country. Digi customers assume full responsibility for learning and meeting the required guidelines for each country in their distribution market. For more information relating to European compliance of an OEM product incorporating the XBee RF Module, contact Digi, or refer to the following web sites:

CEPT ERC 70-03E - Technical Requirements, European restrictions and general requirements: Available at www.ero.dk/.

R&TTE Directive - Equipment requirements, placement on market: Available at www.ero.dk/.

XBee RF Module

The following antennas have been tested and approved for use with the embedded XBee RF Module:

- Dipole (2.1 dBi, Omni-directional, Articulated RPSMA, Digi part number A24-HABSM)
- PCB Antenna (0.0 dBi)
- Monopole Whip (1.5 dBi)

Canada (IC)

Labeling Requirements

Labeling requirements for Industry Canada are similar to those of the FCC. A clearly visible label on the outside of the final product enclosure must display the following text:

Contains Model XBee Radio, IC: 1846A-XBS2C

The integrator is responsible for its product to comply with IC ICES-003 & FCC Part 15, Sub. B - Unintentional Radiators. ICES-003 is the same as FCC Part 15 Sub. B and Industry Canada accepts FCC test report or CISPR 22 test report for compliance with ICES-003.

If it contains an XBee-PRO RF Module, the clearly visible label on the outside of the final product enclosure must display the following text:

Contains Model XBee PRO Radio, IC: 1846A-XBPS2C

Transmitters for Detachable Antennas

This device has been designed to operate with the antennas listed in the previous table and having a maximum of 19 dB. Antennas not included in this list or having a gain greater than 19 dB are strictly prohibited for use with this device. The required antenna impedance is 50 ohms.

Detachable Antenna

To reduce potential radio interference to other users, the antenna type and gain should be so chosen that the equivalent, istropically radiated power (EIRP) is not more than permitted for successful communication.

Australia (C-Tick)

These modules comply with requirements to be used in end products in Australia. All products with EMC and radio communications must have a registered C-Tick mark. Registration to use the compliance mark will only be accepted from Australian manufacturers or importers, or their agent, in Australia.

In order to have a C-Tick mark on an end product, a company must comply with a or b below.

a. have a company presence in Australia.

b. have a company/distributor/agent in Australia that will sponsor the importing of the end product.

Contact Digi for questions related to locating a contact in Australia.

Appendix B: Migrating from XBee ZB to XBee ZB SMT

The XBee ZB SMT modules are designed to be compatible with the XBee ZB. The ZB SMT modules have all the features of the ZB modules, and offer the increased feature set described in this user's guide. For further information on the ZB, see the XBee®/XBee-PRO® ZB RF Modules User's Guide available at www.digi.com.

Pin Mapping

Mapping of the ZB SMT pins to the ZB pins is shown in the table below. The pin names are from the S2C SMT module.

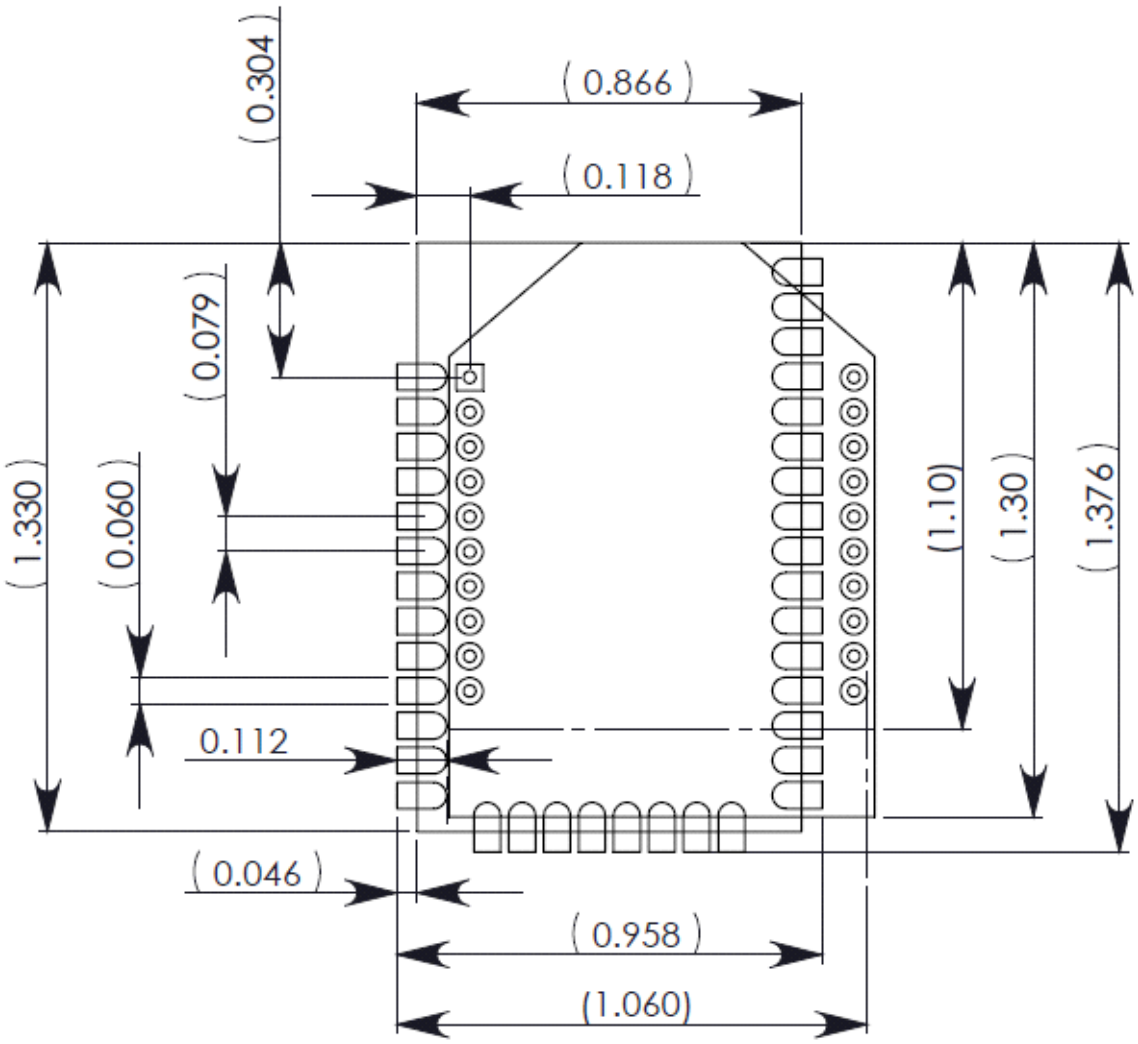
ZB SMT Pin #	Name	ZB Pin #
1	GND	
2	VCC	1
3	DOUT / DIO13	2
4	DIN / $\overline{\text{CONFIG}}$ / DIO14	3
5	DIO12	4
6	$\overline{\text{RESET}}$	5
7	RSSI PWM / DIO10	6
8	PWM1 / DIO11	7
9	[reserved]	8
10	$\overline{\text{DTR}}$ / SLEEP_RQ / DIO8	9
11	GND	10
12	SPI_$\overline{\text{ATTN}}$ / $\overline{\text{BOOTMODE}}$ / DIO19	
13	GND	
14	SPI_CLK / DIO18	
15	SPI_$\overline{\text{SSEL}}$ / DIO17	
16	SPI_MOSI / DIO16	
17	SPI_MISO / DIO15	
18	[reserved]	
19	[reserved]	
20	[reserved]	
21	[reserved]	
22	GND	
23	[reserved]	
24	DIO4	11
25	$\overline{\text{CTS}}$ / DIO7	12
26	ON / $\overline{\text{SLEEP}}$ / DIO9	13
27	VREF	14

ZB SMT Pin #	Name	ZB Pin #
28	ASSOCIATE / DIO5	15
29	$\overline{\text{RTS}}$ / DIO6	16
30	AD3 / DIO3	17
31	AD2 / DIO2	18
32	AD1 / DIO1	19
33	AD0 / DIO0	20
34	[reserved]	
35	GND	
36	RF	
37	[reserved]	

Mounting

One of the important differences between the ZB SMT and the ZB modules is the way they mount to the PCB. The ZB is designed with through-hole pins, while the ZB SMT is designed with Surface Mount Technology (SMT). As such, different mounting techniques are required.

Digi International has designed a footprint which will allow either module to be attached to a PCB. The layout is shown below. All dimensions are in inches.



The round holes in the diagram are for the ZB through-hole design, and the semi-oval pads are for the ZB SMT design. Pin 1 of the through-hole design is lined up with pin 1 of the ZB SMT design, but the pins are actually offset by one pad (see Pin Mapping above). By using diagonal traces to connect the appropriate pins, the layout will work for both modules.

Information on attaching the ZB SMT module is included in Appendix C below.

Appendix C: Manufacturing Information

The XBee is designed for surface mount on the OEM PCB. It has castellated pads to allow for easy solder attach inspection. The pads are all located on the edge of the module, so that there are no hidden solder joints on these modules.

Recommended Solder Reflow Cycle

The recommended solder reflow cycle is shown below. The chart shows the temperature setting and the time to reach the temperature. The cooling cycle is not shown.

Time (seconds)	Temperature (degrees C)
30	65
60	100
90	135
120	160
150	195
180	240
210	260

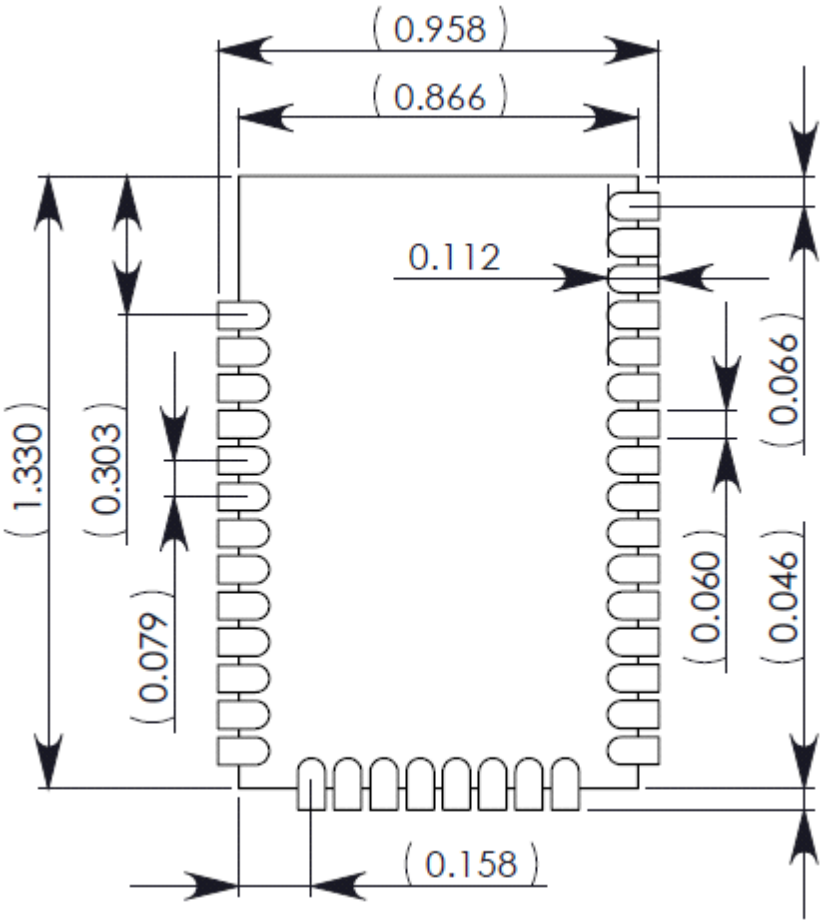
The maximum temperature should not exceed 260 degrees Celsius.

The module will reflow during this cycle, and therefore must not be reflowed updside down. Care should be taken not to jar the module while the solder is molten, as parts inside the module can be removed from their required locations.

Hand soldering is possible and should be done in accordance with approved standards.

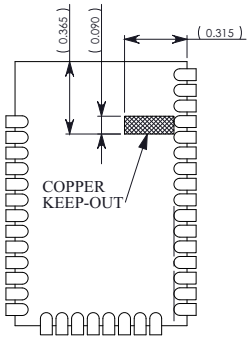
Recommended Footprint

It is recommended that you use the PCB footprint shown below for surface mounting. Dimensions are in inches.



The solder footprint should be matched to the copper pads, but may need to be adjusted depending on the specific needs of assembly and product standards.

While the underside of the module is mostly coated with solder resist, it is recommended that the copper layer directly below the module be left open to avoid unintended contacts. Copper or vias must not interfere with the three exposed RF test points on the bottom of the module (see below). Furthermore, these modules have a ground plane in the middle on the back side for shielding purposes, which can be affected by copper traces directly below the module.



Flux and Cleaning

It is recommended that a “no clean” solder paste be used in assembling these modules. This will eliminate the clean step and ensure unwanted residual flux is not left under the module where it is difficult to remove. In addition:

- Cleaning with liquids can result in liquid remaining under the shield or in the gap between the module and the OEM PCB. This can lead to unintended connections between pads on the module.
- The residual moisture and flux residue under the module are not easily seen during an inspection process.

Factory recommended best practice is to use a “no clean” solder paste to avoid the issues above and ensure proper module operation.

Reworking

Rework should never be performed on the module itself. The module has been optimized to give the best possible performance, and reworking the module itself will void warranty coverage and certifications. We recognize that some customers will choose to rework and void the warranty; the following information is given as a guideline in such cases to increase the chances of success during rework, though the warranty is still voided.

The module may be removed from the OEM PCB by the use of a hot air rework station, or hot plate. Care should be taken not to overheat the module. During rework, the module temperature may rise above its internal solder melting point and care should be taken not to dislodge internal components from their intended positions.

Appendix D: Warranty Information

XBee RF Modules from Digi International, Inc. (the “Product”) are warranted against defects in materials and workmanship under normal use, for a period of 1-year from the date of purchase. In the event of a product failure due to materials or workmanship, Digi will repair or replace the defective product. For warranty service, return the defective product to Digi International, shipping prepaid, for prompt repair or replacement.

The foregoing sets forth the full extent of Digi International’s warranties regarding the Product. Repair or replacement at Digi International’s option is the exclusive remedy. THIS WARRANTY IS GIVEN IN LIEU OF ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, AND DIGI SPECIFICALLY DISCLAIMS ALL WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL DIGI, ITS SUPPLIERS OR LICENSORS BE LIABLE FOR DAMAGES IN EXCESS OF THE PURCHASE PRICE OF THE PRODUCT, FOR ANY LOSS OF USE, LOSS OF TIME, INCONVENIENCE, COMMERCIAL LOSS, LOST PROFITS OR SAVINGS, OR OTHER INCIDENTAL, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PRODUCT, TO THE FULL EXTENT SUCH MAY BE DISCLAIMED BY LAW. SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES. THEREFORE, THE FOREGOING EXCLUSIONS MAY NOT APPLY IN ALL CASES. This warranty provides specific legal rights. Other rights which vary from state to state may also apply.

Appendix E: Definitions

Definitions

ZigBee Node Types	
Coordinator	A node that has the unique function of forming a network. The coordinator is responsible for establishing the operating channel and PAN ID for an entire network. Once established, the coordinator can form a network by allowing routers and end devices to join to it. Once the network is formed, the coordinator functions like a router (it can participate in routing packets and be a source or destination for data packets). -- One coordinator per PAN -- Establishes/Organizes PAN -- Can route data packets to/from other nodes -- Can be a data packet source and destination -- Mains-powered Refer to the XBee coordinator section for more information.
Router	A node that creates/maintains network information and uses this information to determine the best route for a data packet. A router must join a network before it can allow other routers and end devices to join to it. A router can participate in routing packets and is intended to be a mains-powered node. -- Several routers can operate in one PAN -- Can route data packets to/from other nodes -- Can be a data packet source and destination -- Mains-powered Refer to the XBee router section for more information.
End device	End devices must always interact with their parent to receive or transmit data. (See 'joining definition.) They are intended to sleep periodically and therefore have no routing capacity. An end device can be a source or destination for data packets but cannot route packets. End devices can be battery-powered and offer low-power operation. -- Several end devices can operate in one PAN -- Can be a data packet source and destination -- All messages are relayed through a coordinator or router -- Lower power modes

ZigBee Protocol

PAN	Personal Area Network - A data communication network that includes a coordinator and one or more routers/end devices.
Joining	The process of a node becoming part of a ZigBee PAN. A node becomes part of a network by joining to a coordinator or a router (that has previously joined to the network). During the process of joining, the node that allowed joining (the parent) assigns a 16-bit address to the joining node (the child).
Network Address	The 16-bit address assigned to a node after it has joined to another node. The coordinator always has a network address of 0.
Operating Channel	The frequency selected for data communications between nodes. The operating channel is selected by the coordinator on power-up.
Energy Scan	A scan of RF channels that detects the amount of energy present on the selected channels. The coordinator uses the energy scan to determine the operating channel.
Route Request	Broadcast transmission sent by a coordinator or router throughout the network in attempt to establish a route to a destination node.
Route Reply	Unicast transmission sent back to the originator of the route request. It is initiated by a node when it receives a route request packet and its address matches the Destination Address in the route request packet.
Route Discovery	The process of establishing a route to a destination node when one does not exist in the Routing Table. It is based on the AODV (Ad-hoc On-demand Distance Vector routing) protocol.
ZigBee Stack	ZigBee is a published specification set of high-level communication protocols for use with small, low-power modules. The ZigBee stack provides a layer of network functionality on top of the 802.15.4 specification. For example, the mesh and routing capabilities available to ZigBee solutions are absent in the 802.15.4 protocol.