XBee® 868LP

Radio Frequency (RF) Modules

User Guide

Revision history—90002126

Revision	Date	Description
R	October 2016	Converted to the new MadCap Flare format with minor updates and added the information from the XBee 868LP Getting Started Guide (90002127).
S	June 2017	Modified regulatory and certification information as required by RED (Radio Equipment Directive).
T	May 2018	Added note on range estimation.
U	March 2019	Added a receiver category to Performance specifications.
V	July 2021	Added safety instructions and UKCA labeling requirements.

Trademarks and copyright

Disclaimers

Information in this document is subject to change without notice and does not represent a commitment on the part of Digi International. Digi provides this document "as is," without warranty of any kind, expressed or implied, including, but not limited to, the implied warranties of fitness or merchantability for a particular purpose. Digi may make improvements and/or changes in this manual or in the product(s) and/or the program(s) described in this manual at any time.

Warranty

To view product warranty information, go to the following website:

www.digi.com/howtobuy/terms

Customer support

Gather support information: Before contacting Digi technical support for help, gather the following information:

Product name and model

Product serial number (s)

Firmware version

Operating system/browser (if applicable)

Logs (from time of reported issue)

Trace (if possible)

Description of issue

Steps to reproduce

Contact Digi technical support: Digi offers multiple technical support plans and service packages. Contact us at +1 952.912.3444 or visit us at www.digi.com/support.

Feedback

To provide feedback on this document, email your comments to

techcomm@digi.com

Include the document title and part number (XBee 868LP RF Modules User Guide, 90002126 V) in the subject line of your email.

Contents

XBee 868LP RF Modules User Guide

Technical specifications

Hardware

Get started

Configure the XBee 868LP RF Module

Operation

Modes

Sleep modes

Advanced application features

Networking methods

AT commands

Operate in API mode

Migrate from XBee through-hole to surface-mount devices

Manufacturing information

Regulatory information

XBee 868LP RF Modules User Guide

The Digi XBee 868LP RF Modules provide wireless connectivity to end-point devices in mesh networks. With the XBee, users can have their network up-and-running in a matter of minutes without configuration or additional development. The Digi XBee 868LP RF Module consists of firmware loaded onto Digi XBee S8 hardware.

You can build networks up to 128 nodes using the XBee modules. For larger networks up to 1000+ nodes, Digi offers RF Optimization Services to assist with proper network configuration. Contact Digi Technical Support for more details.

Note The Digi XBee 868LP RF Modules are not compatible with other XBee products.

XBee S8 hardware description

The XBee S8 radio module hardware consists of an Energy Micro EFM32G230F128 microcontroller, an Analog Devices ADF7023 radio transceiver, and in the Programmable version, a NXP MC9S08QE32 microcontroller.

European acceptance

The Digi XBee 868LP is manufactured under ISO 900:2015 registered standards.

The Digi XBee 868LP RF Modules are optimized for use in Europe and other regions. For more information, see Regulatory information.

Safety instructions

XBee modules

- The XBee radio module cannot be guaranteed operation due to the radio link and so should not be used for interlocks in safety critical devices such as machines or automotive applications.
- The XBee radio module have not been approved for use in (this list is not exhaustive):
  - medical devices
  - nuclear applications
  - explosive or flammable atmospheres
- There are no user serviceable components inside the XBee radio module. Do not remove the shield or modify the XBee in any way. Modifications may exclude the module from any warranty and can cause the XBee radio to operate outside of regulatory compliance for a given country, leading to the possible illegal operation of the radio.
- Use industry standard ESD protection when handling the XBee module.
- Take care while handling to avoid electrical damage to the PCB and components.
- Do not expose XBee radio modules to water or moisture.
- Use this product with the antennas specified in the XBee module user guides.
- The end user must be told how to remove power from the XBee radio module or to locate the antennas 20 cm from humans or animals.

Technical specifications

Performance specifications

The following table describes the performance specifications for the devices.

Note Range figure estimates are based on free-air terrain with limited sources of interference. Actual range will vary based on transmitting power, orientation of transmitter and receiver, height of transmitting antenna, height of receiving antenna, weather conditions, interference sources in the area, and terrain between receiver and transmitter, including indoor and outdoor structures such as walls, trees, buildings, hills, and mountains.

<table>
<tr><th>Specification</th><th colspan="3">XBee</th></tr>
<tr><td>Indoor/urban range</td><td colspan="3">Up to 370 ft (112 m) with a 2.1 dBi antenna, up to 46 ft (14 m) with a PCB embedded antenna.</td></tr>
<tr><td>Outdoor RF line-of-sight range</td><td colspan="3">Up to 5.2 miles (8.4 km) with a 2.1 dBi antenna, up to 0.4 miles (.64 km) with a PCB embedded antenna.</td></tr>
<tr><td>Transmit power output</td><td colspan="3">Up to 14 dBm (25 mW) EIRP with 2.1 dBi antenna</td></tr>
<tr><td>RF data rate (high)</td><td colspan="3">80 kb/s</td></tr>
<tr><td>RF data rate (low)</td><td colspan="3">10 kb/s</td></tr>
<tr><td>UART interface</td><td colspan="3">Complementary metal–oxide–semiconductor (CMOS) serial universal asynchronous receiver/transmitter (UART), baud rate stability of <1%.</td></tr>
<tr><td>UART data rate (software selectable)</td><td colspan="3">9600-230400 baud</td></tr>
<tr><td>SPI clock rate</td><td colspan="3">Up to 3.5 MHz</td></tr>
<tr><td>Receiver category</td><td colspan="3">Class 2</td></tr>
<tr><td>Receiver sensitivity (typical)</td><td colspan="3">-101 dBm @ 80 kb/s, -106 dBm @ 10 kb/s.</td></tr>
<tr><td rowspan="4">Receiver blocking (typical)</td><td>Frequency offset</td><td colspan="2">Data rate</td></tr>
<tr><td></td><td>10 kb/s</td><td>80 kb/s</td></tr>
<tr><td>+/- 400 kHz</td><td>40 dB</td><td>35 dB</td></tr>
<tr><td>+/- 200 kHz</td><td>35 dB</td><td>29 dB</td></tr>
</table>

Note To determine your indoor/urban range or outdoor RF line-of-sight range, perform a range test under your operating conditions.

LBT and AFA specifications

The following table provides the Listen Before Talk (LBT) and Adaptive Frequency Agility (AFA) specifications.

Specification	**XBee 868LP**
Channel spacing	100 kHz
Receiver bandwidth	150 kHz
Modulation bandwidth	< 300 kHz
LBT threshold	< -88 dBm
TX on time	< 1 second

Power requirements

The following table describes the power requirements for the XBee 868LP RF Module.

Specification	**XBee**
Supply voltage (V_{DD})	2.7 to 3.6 VDC
Transmit current, high data rate	48 mA, (45 mA typical)
Transmit current, low data rate	47 mA (41 mA typical)
Idle / receive current (high data rate)	27 mA (22 mA typical)
Idle / receive current (low data rate)	26 mA (24 mA typical)
Sleep current	1.7 µA

General specifications

The following table describes the general specifications for the devices.

Specification	**XBee**
Operating frequency band	863 to 870 MHz for Europe
Dimensions	2.119 x 3.4 x 0.305 cm (0.866 x 1.333 x 1.2 in)
Weight	40 g (1.4 oz)
Operating temperature	-40 ºC to 85 ºC (industrial)
Antenna options	U.FL RF connector, RF pad, embedded PCB antenna. **Note** The embedded PCB antenna is only approved with 10 kb/s data rate, not 80 kb/s data rate.
Digital I/O	13 I/O lines, five dedicated to Serial Peripheral Interface (SPI) that can be used as digital outputs.
ADC	4 10-bit analog inputs

Networking and security

The following table describes the networking and security specifications for the devices.

Specification	XBee
Supported network topologies	Mesh, repeater, point-to-point, point-to-multipoint, peer-to-peer.
Number of channels, user selectable channels	30 channels, LBT + AFA
Addressing options	Personal Area Network identifier (PAN ID) and 64-bit addresses.
Encryption	128 bit Advanced Encryption Standard (AES)

Note For more information about the number of user selectable channels, see CE and UKCA OEM labeling requirements for countries in the European Community.

Regulatory conformity summary

This table describes the agency approvals for the devices.

Specification	XBee
Europe (CE)	Yes

Serial communication specifications

The XBee 868LP RF Module supports both Universal Asynchronous Receiver / Transmitter (UART) and Serial Peripheral Interface (SPI) serial connections.

UART pin assignments

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

For more information on UART operation, see UART data flow.

SPI pin assignments

SPI Pins	Module Pin Number
SPI_SCLK / DIO18 (input)	14
SPI_$\overline{\text{SSEL}}$ / DIO17 (input)	15
SPI_MOSI / DIO16 (input)	16
SPI_MISO / DIO15 (output/tri-stated)	17
SPI_$\overline{\text{ATTN}}$ (output)	12

For more information on SPI operation, see SPI communications.

GPIO specifications

The XBee 868LP RF Modules have General Purpose Input / Output (GPIO) ports available. The exact list depends on the module configuration, as some GPIO pads are used for purposes such as serial communication.

You can set the pin configuration by using D0-D9, P0-P9, and I/O line monitoring. You cannot sample pins P5-P9, but you may use them as outputs. For more information on these commands, see AT commands. For more information on configuring and using GPIO ports, see Pin signals.

The following table provides the electrical specifications for the GPIO pads.

GPIO electrical specification	Value
Low Schmitt switching threshold	0.3 x V_{DD}
High Schmitt switching threshold	0.7 x V_{DD}
Input pull-up resistor value	40 kΩ
Input pull-down resistor value	40 kΩ
Output voltage for logic 0	0.05 x V_{DD}
Output voltage for logic 1	0.95 x V_{DD}
Output source current	6 mA
Output sink current	6 mA
Total output current (for GPIO pads)	48 mA

Hardware specifications for the programmable variant

If the module includes the programmable secondary processor, add the following table values to the specifications listed in Pin signals, Serial communication specifications, and GPIO specifications. For example, if the secondary processor is running at 20 MHz and the primary processor is in receive mode, then the new current value will be *Itotal = Ir2 + Irx = 14 mA + 9 mA = 23 mA*, where *Ir2* is the runtime current of the secondary processor and Irx is the receive current of the primary.

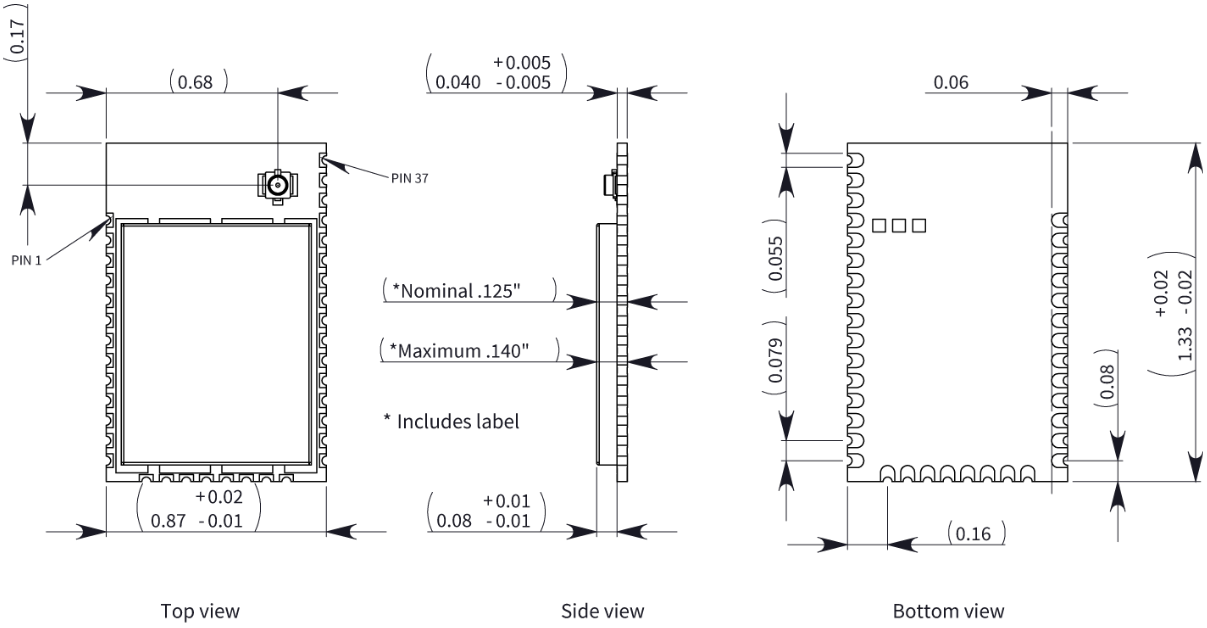
The following table provides the specifications of the programmable secondary processor.

Optional secondary processor specification	Add to RX, TX, and sleep currents specifications depending on mode of operation
Runtime current for 32 k running at 20 MHz	+14 mA
Runtime current for 32 k running at 1 MHz	+1 mA
Sleep current	+0.5 µA typical
V_{REF} Range	1.8 VDC to V_{DD}
Microcontroller	NXP Flexis 8-bit S08 microcontroller NXP S08QE Family Part number: MC9S08QE32

Hardware

Mechanical drawings

The following mechanical drawings of the XBee 868LP RF Modules show all dimensions in inches. Antenna options are not shown.



Top view

Side view

Bottom view

Pin signals

The following table describes the pin assignments for the devices. A horizontal line above the signal name indicates low-asserted signals.

Pin#	Name	Direction	Default state	Description
1	GND	-	-	Ground
2	V_{DD}	-	-	Power supply
3	DIO13 / DOUT	Both	Output	GPIO/UART Data Out
4	DIO14 / DIN / $\overline{\text{CONFIG}}$	Both	Input	GPIO/UART Data In
5	DIO12	Both		GPIO
6	$\overline{\text{RESET}}$	Input		Module reset. Drive low to reset the module. This is also an output with an open drain configuration with an internal 20 kW pull-up (never drive to logic high, as the module may be driving it low). The minimum pulse width is 1

Pin#	Name	Direction	Default state	Description
				mS.
7	DIO10 / RSSI PWM0	Both	Output	GPIO/RX Signal Strength Indicator
8	DIO11 / PWM1	Both	Disabled	GPIO/Pulse Width Modulator
9	[reserved]	-	Disabled	Do not connect
10	DIO8 / SLEEP_REQUEST	Both	Input	GPIO/Pin Sleep Control Line (DTR on the dev board)
11	GND	-	-	Ground
12	DIO19 / SPI_ATTN	Output	Output	Serial Peripheral Interface Attention or UART Data Present indicator
13	GND	-	-	Ground
14	DIO18 / SPI_CLK	Input	Input	GPIO/Serial Peripheral Interface Clock/
15	DIO17 / SPI_SSEL/	Input	Input	GPIO/Serial Peripheral Interface not Select
16	DIO16 / SPI_MOSI	Input	Input	GPIO/Serial Peripheral Interface Data In
17	DIO15 / SPI_MISO/	Output	Output	GPIO/Serial Peripheral Interface Data Out Tri-stated when SPI_SSEL is high
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	DIO7 / CTS/	Both	Output	GPIO/Clear to Send Flow Control

Pin#	Name	Direction	Default state	Description
26	ON/SLEEP/DIO9	Both	Output	GPIO/Module Status Indicator
27	VREF	Input	-	Not used internally. 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	DIO5 / ASSOCIATE/	Both	Output	GPIO/Associate Indicator
29	DIO6 / RTS	Both	Input	GPIO/Request to Send Flow Control
30	DIO3 / AD3	Both	Disabled	GPIO/Analog Input
31	DIO2 / AD2	Both	Disabled	GPIO/Analog Input
32	DIO1 / AD1	Both	Disabled	GPIO/Analog Input
33	DIO0 / AD0	Both	Input	GPIO/Analog Input
34	[reserved]	-	Disabled	Do not connect
35	GND	-	-	Ground
36	RF	Both	-	RF I/O for RF Pad Variant
37	[reserved]	-	Disabled	Do not connect
Signal Direction is specified with respect to the device. See Design notes for details on pin connections. * These pins are not available for customer use.				

Design notes

The XBee modules do not require any external circuitry or specific connections for proper operation. However, there are some general design guidelines that we recommend to build and troubleshoot a robust design.

Power supply design

A poor power supply can lead to poor radio performance, especially if you do not keep the supply voltage within tolerance or if the noise is excessive. To help reduce noise, place a 1.0 µF and 47 pF capacitor as near as possible to pin 2 on the PCB. If you are using a switching regulator for the power supply, switch the frequencies above 500 kHz. Limit the power supply ripple to a maximum 250 mV peak to peak.

For designs using the programmable modules, we recommend an additional 10 μF decoupling cap near pin 2 of the device. The nearest proximity to pin 2 of the three caps should be in the following order:

1. 47 pf
2. 1 μF
3. 10 μF

Board layout

We design XBee modules to be self-sufficient and have minimal sensitivity to nearby processors, crystals or other printed circuit board (PCB) components. Keep power and ground traces thicker than signal traces and make sure that they are able to comfortably support the maximum current specifications. There are no other special PCB design considerations to integrate XBee modules, with the exception of antennas.

To view a recommended PCB footprint for the module, see Manufacturing information.

Antenna performance

Antenna location is important for optimal performance. The following suggestions help you achieve optimal antenna performance. Point the antenna up vertically (upright). Antennas radiate and receive the best signal perpendicular to the direction they point, so a vertical antenna's omnidirectional radiation pattern is strongest across the horizon.

Position the antennas away from metal objects whenever possible. Metal objects between the transmitter and receiver can block the radiation path or reduce the transmission distance. Objects that are often overlooked include:

- Metal poles
- Metal studs
- Structure beams
- Concrete, which is usually reinforced with metal rods

If you place the device inside a metal enclosure, use an external antenna. Common objects that have metal enclosures include:

- Vehicles
- Elevators
- Ventilation ducts
- Refrigerators
- Microwave ovens
- Batteries
- Tall electrolytic capacitors

Use the following additional guidelines for optimal antenna performance:

- Do not place XBee modules with the chip antenna inside a metal enclosure.
- Do not place any ground planes or metal objects above or below the antenna.

- For the best results, mount the device at the edge of the host PCB. Ensure that the ground, power, and signal planes are vacant immediately below the antenna section.

Recommended pin connections

The only required pin connections for two-way communication are VDD, GND, DOUT and DIN. To support serial firmware updates, you must connect VDD, GND, DOUT, DIN, RTS, and DTR.

Do not connect any pins that are not in use. Use the **PR** and **PD** commands to pull all inputs on the radio high or low with 40k internal pull-up or pull-down resistors. Unused outputs do not require any specific treatment.

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

You can connect other pins to external circuitry for convenience of operation including the Associate LED pad (pad 28) and the Commissioning pad (pad 33). The Associate LED pad flashes 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. For more information see Commissioning pushbutton and associate LED. The source and sink capabilities are limited to 6 mA on all I/O pads.

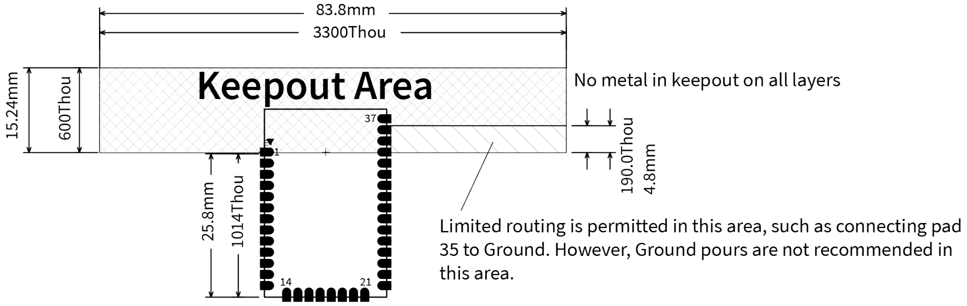
Only the programmable versions of these devices use the VREF pad (pad 27). For compatibility with other XBee modules, we recommend connecting this pin to a voltage reference if you want to enable analog sampling. Otherwise, connect to GND.

Design notes for PCB antenna devices

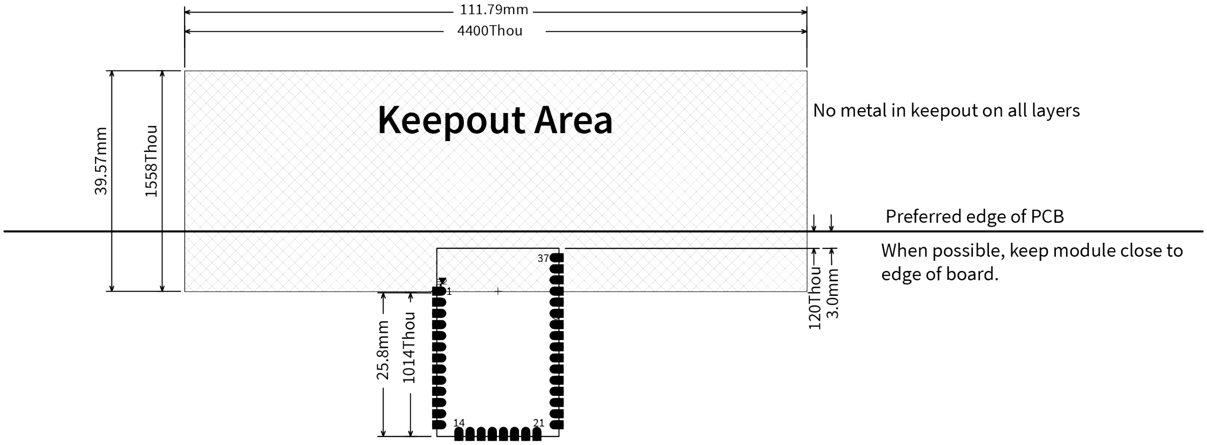
Position PCB antenna devices so there are no ground planes or metal objects above or below the antenna. For best results, do not place the device in a metal enclosure, as this may greatly reduce the range. Place the device at the edge of the PCB on which it is mounted. Make sure the ground, power and signal planes are vacant immediately below the antenna section.

The following drawings illustrate important recommendations when you are designing with PCB antenna devices. For optimal performance, do not mount the device 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)


83.8mm
3300Thou
Keepout Area
No metal in keepout on all layers
15.24mm
600Thou
37
1
25.8mm
1014Thou
14
21
190.0Thou
4.8mm
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)



The antenna performance improves with a larger keepout area

Notes

1. We recommend non-metal enclosures. For metal enclosures, use an external antenna.
2. Keep metal chassis or mounting structures in the keepout area at least 2.54 cm (1 in) from the antenna.
3. Maximize the distance between the antenna and metal objects that might be mounted in the 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.

Design notes for RF pad devices

The RF pad is a soldered antenna connection. The RF signal travels from pin 33 on the device to the antenna through an RF trace transmission line on the PCB. Any additional components between the device and antenna violates modular certification. The controlled impedance for the RF trace is 50 Ω.

We recommend using a microstrip trace, although you can also use a coplanar waveguide if you need more isolation. A microstrip generally requires less area on the PCB than a coplanar waveguide. We do not recommend using a stripline because sending the signal to different PCB layers can introduce matching and performance problems.

Following good design practices is essential when implementing the RF trace on a PCB. Consider the following points:

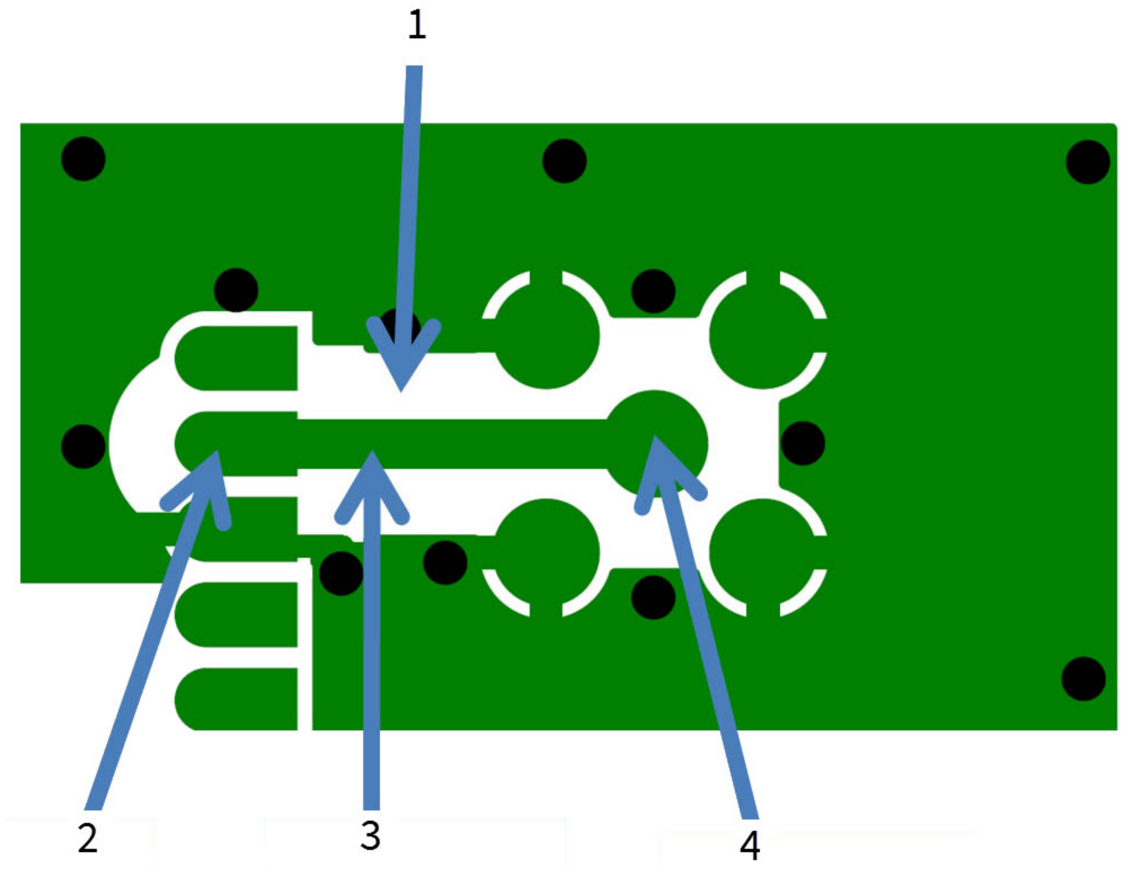
- Minimize the length of the trace by placing the RPSMA jack close to the device.
- Connect all of the grounds on the jack and the device to the ground planes directly or through closely placed vias.
- Space any ground fill on the top layer at least twice the distance **d** (in this case, at least 0.028") from the microstrip to minimize their interaction.

Additional considerations:

- 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 33 of the device to the RF pin of the RPSMA connector.
- The RF trace width determines the impedance of the transmission line with relation to the ground plane. Many online tools can estimate this value, although you should consult the PCB manufacturer for the exact width.

Implementing these design suggestions helps ensure that the RF pad device performs to its specifications.

The following figures show a layout example of a host PCB that connects an RF pad device to a right angle, through-hole RPSMA jack.

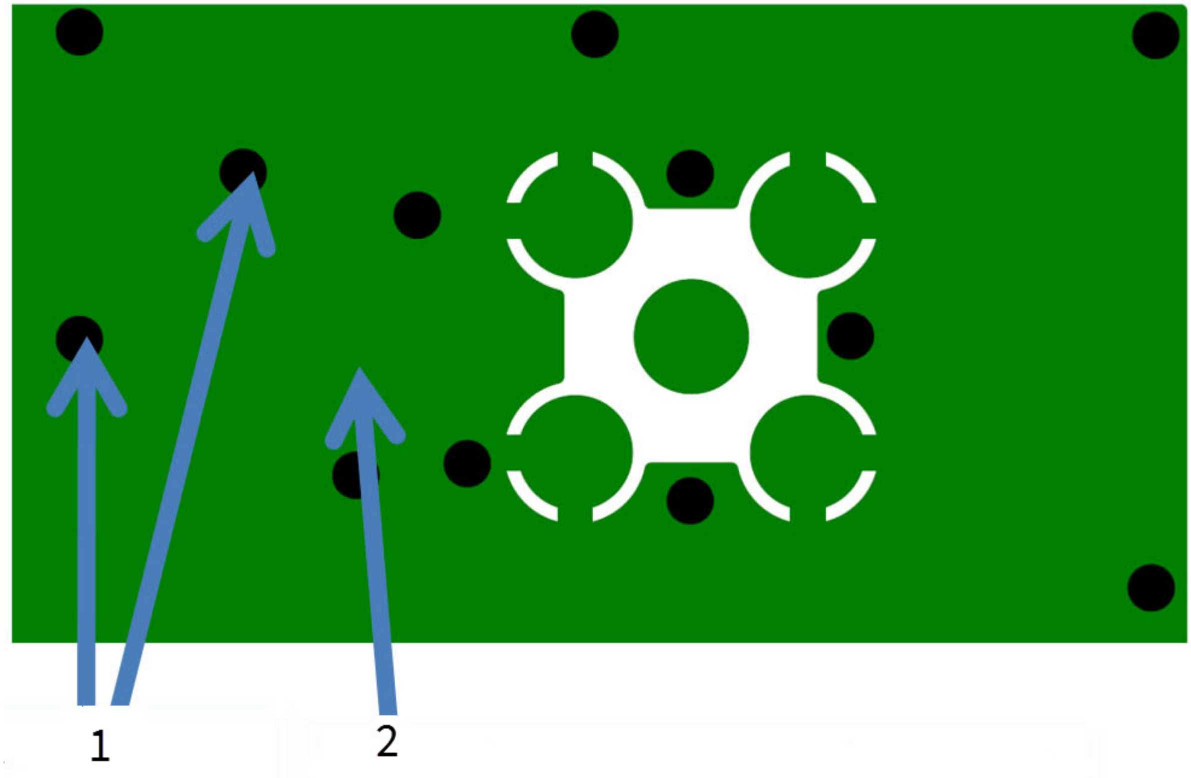


Number	Description
1	Maintain a distance of at least 2 d between microstrip and ground fill.
2	Device pin 33.
2	RF pad pin.
3	50 Ω microstrip trace.
4	RF connection of RPSMA jack.

The width in this example is approximately 0.025 in for a 50 Ω trace, assuming d = 0.014 in, and that the dielectric has a relative permittivity of 4.4. This trace width is a good fit with the device footprint's 0.335" pad width.

Note We do not recommend using a trace wider than the pad width, and using a very narrow trace (under 0.010") can cause unwanted RF loss.

The following illustration shows PCB layer 2 of an example RF layout.



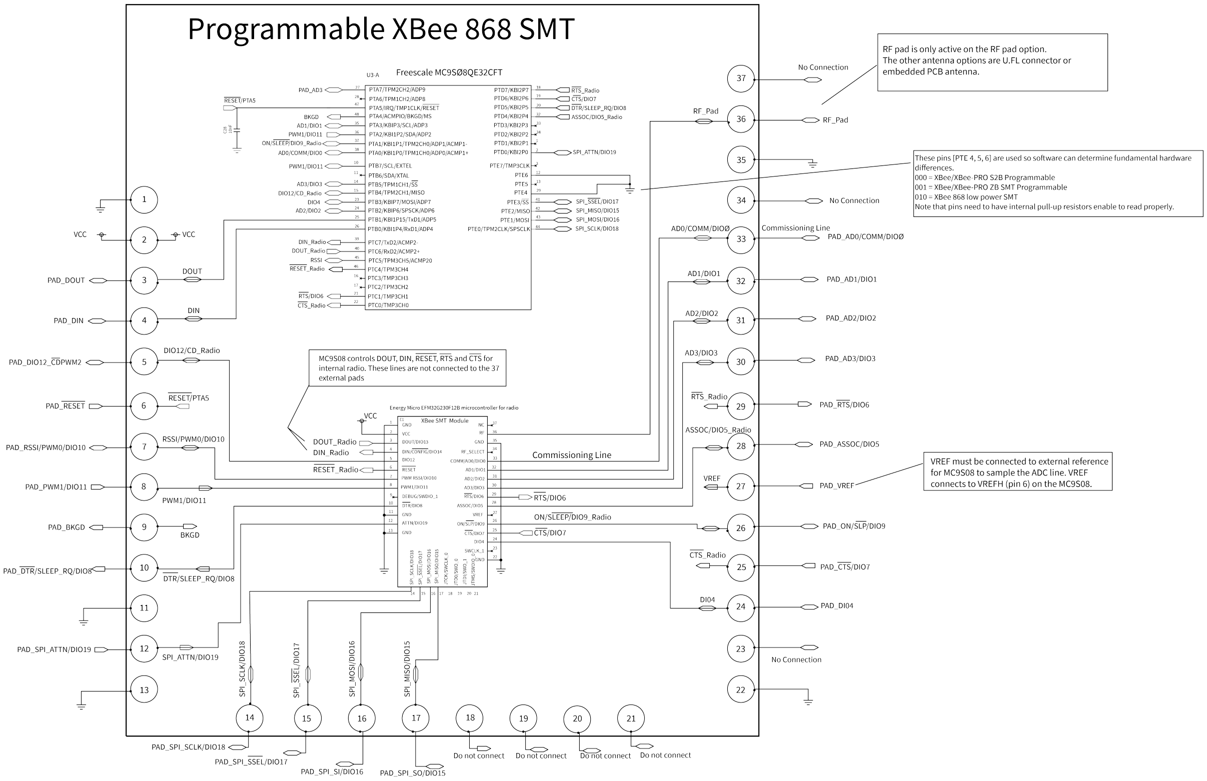
Number	Description
1	Use multiple vias to help eliminate ground variations.
2	Put a solid ground plane under RF trace to achieve the desired impedance.

Module operation for the 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 internal microcontroller or the MC9SO8QE micro; see the block diagram in Operation for details. The internal microcontroller by default has control of certain lines. The internal microcontroller can release these lines by sending the proper command(s) to disable the desired DIO line(s). For more information about commands, see AT commands.

For the secondary processor to sample with ADCs, the XBee 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 .


Programmable XBee 868 SMT

RF pad is only active on the RF pad option.
The other antenna options are U.FL connector or embedded PCB antenna.

These pins [PTE 4, 5, 6] are used so software can determine fundamental hardware differences.
000 = XBee/XBee-PRO S2B Programmable
001 = XBee/XBee-PRO ZB SMT Programmable
010 = XBee 868 low power SMT
Note that pins need to have internal pull-up resistors enable to read properly.

MC9S08 controls DOUT, DIN, RESET, RTS and CTS for internal radio. These lines are not connected to the 37 external pads

VREF must be connected to external reference for MC9S08 to sample the ADC line. VREF connects to VREFH (pin 6) on the MC9S08.

Programmable XBee SDK

The XBee Programmable module is equipped with a NXP MC9S08QE32 application processor. This application processor comes with a supplied bootloader. To interface your application code running on this processor to the XBee Programmable module's supplied bootloader, use the Programmable XBee SDK.

To use the SDK, you must also download CodeWarrior. The download links are:

- CodeWarrior IDE: http://ftp1.digi.com/support/sampleapplications/40003004_B.exe
- Programmable XBee SDK: http://ftp1.digi.com/support/sampleapplications/40003003_D.exe

If these revisions change, search for the part number on Digi’s website. For example, search for **40003003**.

Install the IDE first, and then install the SDK.

The documentation for the Programmable XBee SDK is built into the SDK, so the Getting Started guide appears when you open CodeWarrior.

Get started

The XBee 868LP RF Modules support low-power, peer-to-peer or wireless mesh networks for Europe (868 MHz). The XBee 868LP RF Modules provide reliable delivery of data between remote devices.

This guide shows you how to set up a mesh network using the DigiMesh protocol, send data between devices, and adjust XBee 868LP RF Module settings.

Note For more information about DigiMesh protocol and features, see DigiMesh networking.

This guide covers the following tasks and features:

Set up the devices

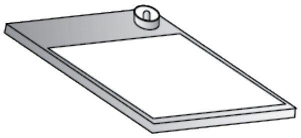
Before you begin

To get started with your XBee RF module development kit, verify that your kit has all of the components and that you meet the system requirements.

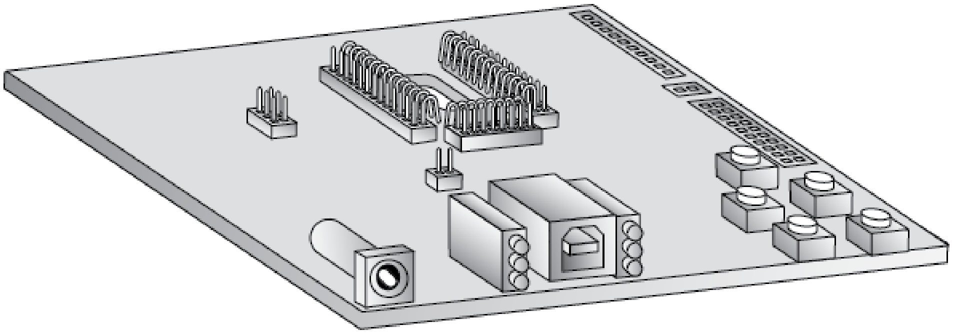
Verify kit contents

The XBee 868LP RF Module development kit contains the following components:

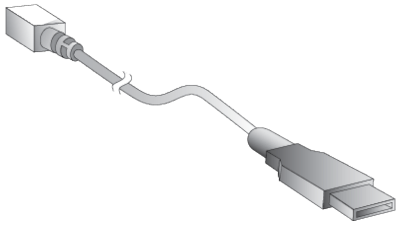
XBee
U.FL module (3)



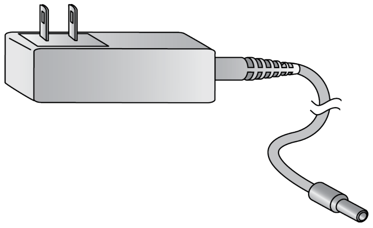
XBee
development
board (3)



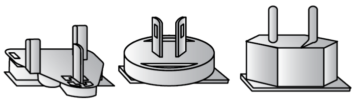
USB cable (2)



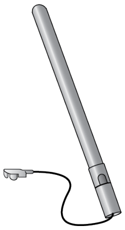
Power supply
(2)



Set of power
supply adapters
(2)



Antenna U.FL (3)



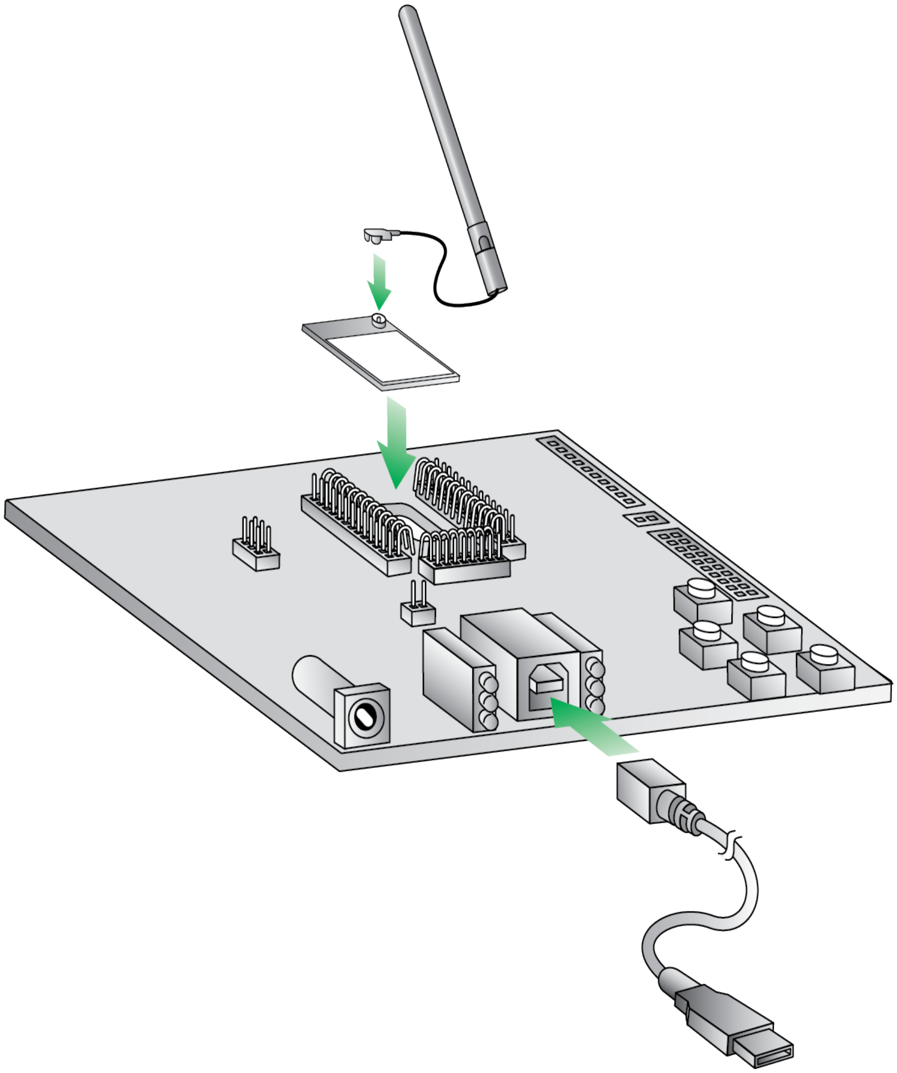
Gather required materials

To complete the steps in this guide, you need the following items:

Item	Description
Computer	Operating systems: ■ Windows Vista/7/8 (32-bit or 64-bit versions) ■ Mac OS X v10.6 and higher versions (64-bit only) ■ Linux with KDE or GNOME window managers (32-bit or 64-bit versions) System requirements: ■ HDD space: 500 MB minimum, 1GB recommended ■ RAM memory: 2 GB minimum, 4 GB recommended ■ CPU: Dual-core processor minimum, Quad-core processor recommended USB ports: ■ Three available USB ports for the XBee/XBee-PRO DigiMesh 2.4 development kit ■ Two available USB ports for the XBee 868LP development kit **Note** Only one computer is required to follow along with the steps in this guide. However, you can use two or more computers—one for each XBee module. For range testing, we recommend a laptop.
XCTU software	Version 6.1.3 or later. See Download and install XCTU.
USB drivers	Windows Vista and later: USB drivers automatically install through plug-and-play. Windows XP and earlier: You need to download the driver software. See Optional: Manually install USB drivers.

Connect the hardware

The following illustration shows you how to assemble the hardware components of the development kit.



1. Attach the XBee 868LP RF Modules to the development boards.
2. Attach the antennas to the devices.
3. Connect the USB cables to the development boards.

CAUTION! Before you remove a device from a development board, make sure the board is not powered by a USB cable or a battery.

Step 1: Download and install XCTU

This section contains download and install instructions based on operating system. XCTU is compatible with Linux, OSX, and Windows. It may be necessary to configure your system prior to installing XCTU for the first time.

If you get stuck, see XCTU installation error.

Download and install XCTU - Windows

Follow the steps below to download and install XCTU on your computer.

1. Go to www.digi.com/xctu.
2. Click **Download**.
3. Under **Download XCTU**, click the Windows installer link.
4. Once the download is complete, run the executable file and follow the steps in the XCTU Setup Wizard.

   Once installation is complete, a “What’s new” dialog appears where you can review the new XCTU features.

Download and install XCTU - Linux

By default, access to the serial and USB ports in Linux is restricted to root and dialout group users. To access your XBee devices and use XCTU to communicate with them, it is mandatory that your Linux user belongs to this group. To add your Linux user to the dialout group:

1. Open a terminal console.
2. Execute the following command where <user> is the user you want to add to the dialout group:

```
sudo usermod -a -G dialout <user>
```

3. Log out and log in again with your user in the system.
4. Go to www.digi.com/xctu.
5. Click **Download**.
6. Under **Download XCTU**, click the Linux installer link.
7. Once the download is complete, run the executable file and follow the steps in the XCTU Setup Wizard.

   Once installation is complete, a “What’s new” dialog appears where you can review the new XCTU features.

Download and install XCTU - OSX

OSX version 10.8 (Mountain Lion) and greater only allows you to install applications downloaded from the Apple Store. To install XCTU, you must temporarily disable this setting. Follow these steps to enable installation of "unsigned" software:

1. Click the Apple icon in the top-left corner of your screen and choose **System Preferences**.
2. Click the **Security & Privacy** icon.
3. To edit security settings, click the padlock icon in the bottom left of the window.
4. Enter your Mac credentials and click **Unlock**. The **Allow applications downloaded from** dialog appears.
5. Click the **Anywhere** radio button and, in the confirmation window, click **Allow From Anywhere**.

Note We recommend you set this option back to **Mac App Store** or **Mac App Store and identified developers** once you have finished installing XCTU.

6. Go to www.digi.com/xctu.
7. Click **Download**.
8. Under **Download XCTU**, click the OSX installer link.
9. Once the download is complete, unzip and run the executable file and follow the steps in the XCTU Setup Wizard.

   Once installation is complete, a “What’s new” dialog appears where you can review the new XCTU features.

Optional: Install XCTU updates

When you start XCTU, you may be notified about software updates. You should always run the latest version of XCTU.

1. When a new version is available, a popup window appears in the bottom-right corner of XCTU.
2. Click on that window and follow the prompts to proceed with the update.

You can also check for updates and manually update the tool by clicking **Help** > **Check for XCTU Updates**.

Optional: Manually install USB drivers

When you connect the XBee board to your computer for the first time, drivers are automatically installed. You can also install device drivers manually:

1. Download and install the appropriate USB drivers from the Digi Support Site.
2. Choose your operating system.
3. Download and run the file.
4. Follow the steps in the installation wizard.

Step 2: Set up your first wireless connection

This section shows you how to configure two XBee modules in AT (transparent) mode. The XBee module passes information along exactly as it receives it. All serial data received by the XBee module is sent wirelessly to a remote destination XBee module.

If you get stuck, see Troubleshooting.

Add devices to XCTU

These instructions show you how to add two devices to XCTU. However, you can use these instructions to add any number of devices.

1. Connect two XBee 868LP RF Modules to your computer using the USB cables.

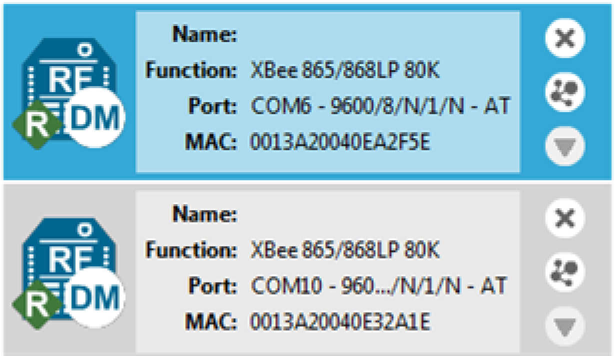
   Tip Connect the two shorter range XBee modules instead of the longer range XBee-PRO modules. This will make it easier to set up a mesh network. See Connect the hardware.

2. Launch XCTU .
3. Click the **Configuration working modes** button ⚙.
4. Click the **Discover radio modules** button .

5. In the **Discover radio devices** dialog, select the serial ports where you want to look for devices and click **Next**.
6. In the Set port parameters window, maintain the default values and click **Finish**.

   As XCTU locates devices, they appear in the **Discovering radio modules** dialog box.
7. Click **Add selected devices** once the discovery process has finished.

   You should see something like this in the **Radio Modules** section:



Name:
Function: XBee 865/868LP 80K
Port: COM6 - 9600/8/N/1/N - AT
MAC: 0013A20040EA2F5E

Name:
Function: XBee 865/868LP 80K
Port: COM10 - 960.../N/1/N - AT
MAC: 0013A20040E32A1E

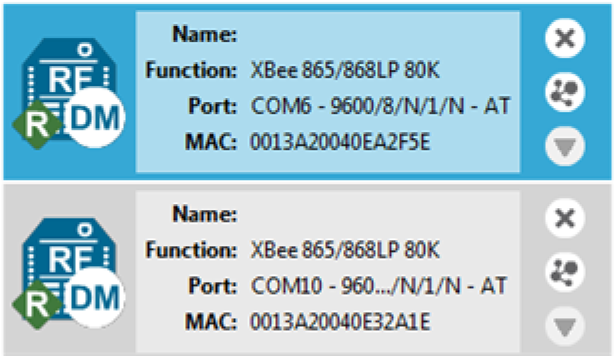
Configure the first two devices in Transparent mode

To transmit data wirelessly between your XBee devices, configure them to be in the same network.

Tip To locate a device, select it in XCTU and click the **Read radio settings** button. The Rx and Tx LED lights on its development board blink green and yellow.

Set up the first XBee device (XBEE_A)

1. Select the first XBee device.



Name:
Function: XBee 865/868LP 80K
Port: COM6 - 9600/8/N/1/N - AT
MAC: 0013A20040EA2F5E

Name:
Function: XBee 865/868LP 80K
Port: COM10 - 960.../N/1/N - AT
MAC: 0013A20040E32A1E

2. Click the **Load default firmware settings** button.

   Tip In the following steps, type parameter letters in the Search box SL to quickly find a parameter.

3. Configure the following parameters:

   ID: 2015
   DH: 0013A200

DL: SL of XBEE_B (Enter the last eight characters of the MAC address for XBEE_B. Or select XBEE_B and find its **SL** value.)
NI: XBEE_A

4. Click the **Write radio settings** button .

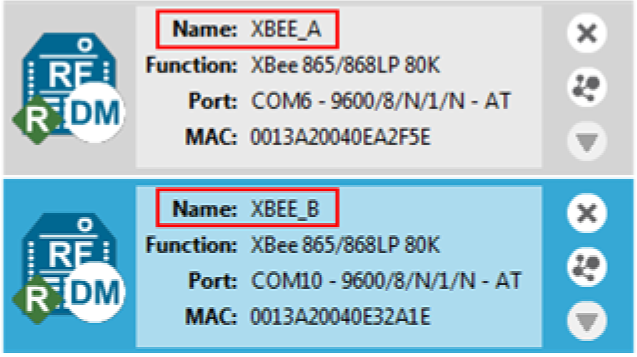
Set up the second XBee device (XBEE_B)

1. Configure the following parameters:

   ID: 2015
   DH: 0013A200
   DL: SL of XBEE_A (Enter the last eight characters of the MAC address for XBEE_A. Or select XBEE_A and find its **SL** value.)
   NI: XBEE_B

2. Click the **Write radio settings** button .

After you write the radio settings for the XBee devices, their names appear in the **Radio Modules** area.



Name: XBEE_A
Function: XBee 865/868LP 80K
Port: COM6 - 9600/8/N/1/N - AT
MAC: 0013A20040EA2F5E

Name: XBEE_B
Function: XBee 865/868LP 80K
Port: COM10 - 9600/8/N/1/N - AT
MAC: 0013A20040E32A1E

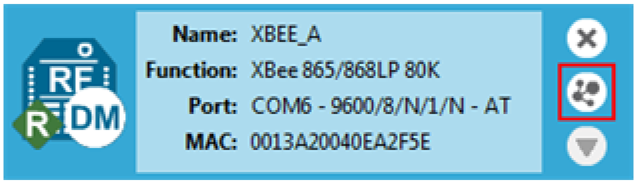
For more information about the parameters, see the following table:

Parameter	XBEE_A	XBEE_B	Effect
ID	2015	2015	Defines the network that a device will attach to. This must be the same for all devices in your network.
DH	0013A200	0013A200	Defines the destination address (high part) for the message.
DL	**SL** of XBEE_B	**SL** of XBEE_A	Defines the destination address (low part) for the message. The value of this setting is the Serial Number Low (**SL**) of the other XBee device.
NI	XBEE_A	XBEE_B	Defines the node identifier. Note The default **NI** value is a blank space. Delete the space when you change the value.

Check the network

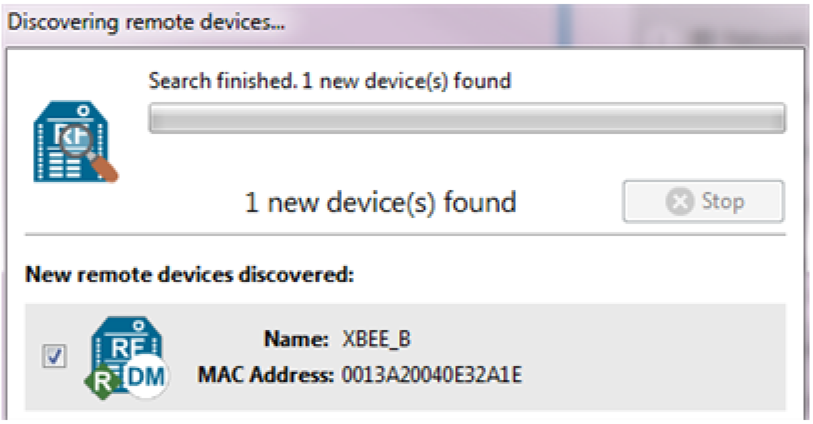
Once both XBee 868LP RF Modules are configured, use XCTU to check that they are in the same network and can see each other.

1. Click the **Discover radio nodes in the same network** button of XBEE_A.



Name: XBEE_A
Function: XBee 865/868LP 80K
Port: COM6 - 9600/8/N/1/N - AT
MAC: 0013A20040EA2F5E

The XBee 868LP RF Module searches for radio modules in the same network.



Discovering remote devices...

Search finished. 1 new device(s) found

1 new device(s) found

New remote devices discovered:

☑ Name: XBEE_B
MAC Address: 0013A20040E32A1E

When the discovery process is finished, XCTU lists discovered devices found within the network in the **Discovering remote devices** dialog.

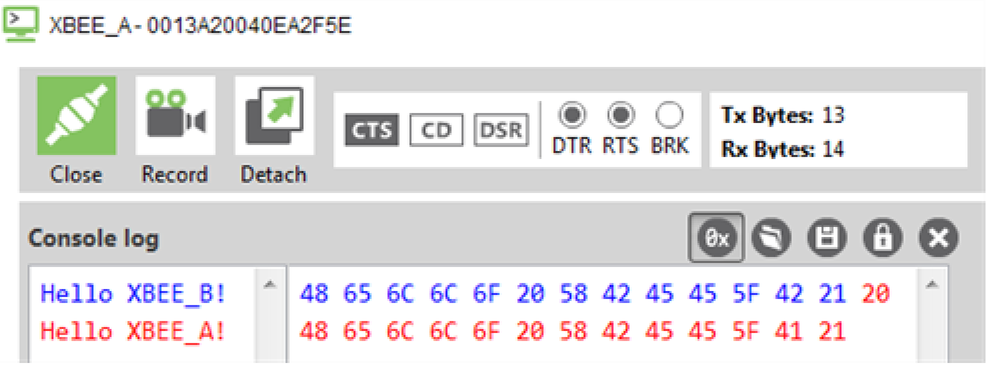
2. Click **Cancel**. There is no need to add the remote device that has been discovered.

Send messages through XCTU

Use the XCTU console to have the two devices send messages to each other.

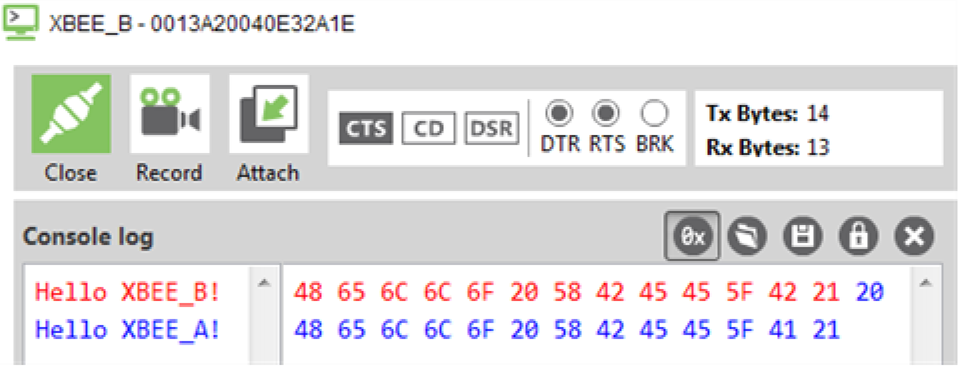
1. Switch both XBee 868LP RF Modules to the consoles working mode .
2. Open a serial connection for each XBee.
   a. Select **XBEE_A** and click .
   b. Select **XBEE_B** and click .
3. Click the **Detach view** button to see both consoles at the same time.
   a. In the **Console log** area for XBEE_A, type "Hello XBEE_B!"
   b. In the **Console log** area for XBEE_B, type "Hello XBEE_A!"

   The message of the sender is in blue font, and the message of the receiver is in red font.



Tx Bytes: 13
Rx Bytes: 14

```
Hello XBEE_B!    48 65 6C 6C 6F 20 58 42 45 45 5F 42 21 20
Hello XBEE_A!    48 65 6C 6C 6F 20 58 42 45 45 5F 41 21
```



Tx Bytes: 14
Rx Bytes: 13

```
Hello XBEE_B!    48 65 6C 6C 6F 20 58 42 45 45 5F 42 21 20
Hello XBEE_A!    48 65 6C 6C 6F 20 58 42 45 45 5F 41 21
```

4. Close the window for XBEE_B.
5. Keep the serial connections open for both XBee modules.

If the two XBee 868LP RF Modules are unable to talk to each other:

- Verify that you accurately configured the parameters. See Configure the first two devices in Transparent mode.
- Verify that the following parameters are configured appropriately:
  - XBee/XBee-PRO DigiMesh 2.4: The CH (Operating Channel) is the same for both XBee modules.
  - XBee 868LP: The CM (Channel Mask) and HP (Preamble ID) are the same for both XBee modules.

Step 3: Create a mesh network

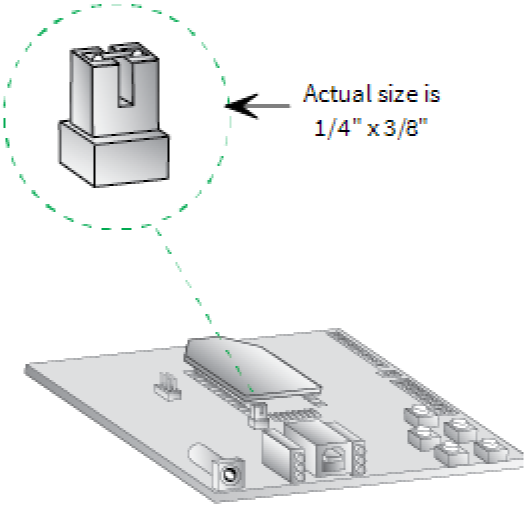
This section describes how to add a third XBee module to create a mesh network. Establish a mesh network any time you want to create a network that is larger than the range of each individual radio. In these instructions, you first connect a loopback jumper to an XBee module in preparation for testing your network.

If you get stuck, see Troubleshooting.

Connect a loopback jumper to an XBee device

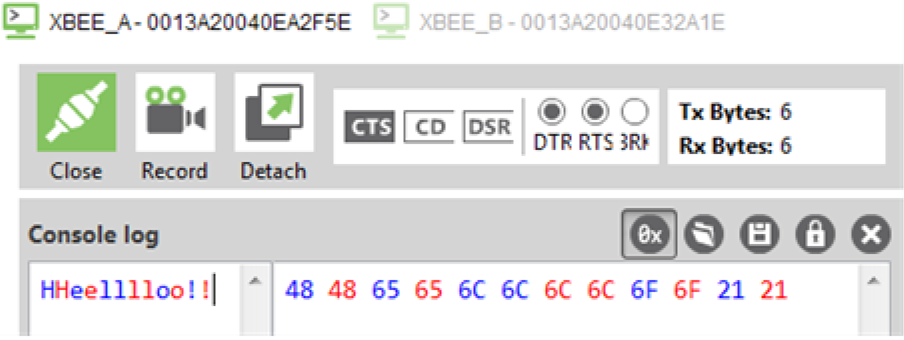
Connecting a loopback jumper to an XBee device lets you send a message to another XBee device and have the message loop back to the sender.

1. Connect the loopback jumper on XBEE_B so it bridges the two pins on its development board.



2. In the XBEE_A console, click the **Clear session** button ⊗ to clear your previous conversation.
3. Type **Hello!**

   Each character loops back in the XBEE_A console log, which indicates that XBEE_A successfully sent the message to XBEE_B.



Tx Bytes: 6
Rx Bytes: 6

```
HHeellllloo!!
48 48 65 65 6C 6C 6C 6C 6F 6F 21 21
```

   You are now ready to use the loopback jumper to help you test a mesh network consisting of three XBee devices.

Set up a third XBee module to create a mesh network

To create a mesh network, move XBEE_B away from XBEE_A until communication is lost; then add XBEE_C to relay messages between XBEE_A and XBEE_B. The network automatically adjusts and redirects communications as soon as a pathway becomes available.

1. Move XBEE_B out of range of XBEE_A:
   a. Disconnect XBEE_B from your computer and remove it from XCTU.
   b. Connect XBEE_B to a power supply (or laptop or portable battery) and move it away from XBEE_A until it is out of range.

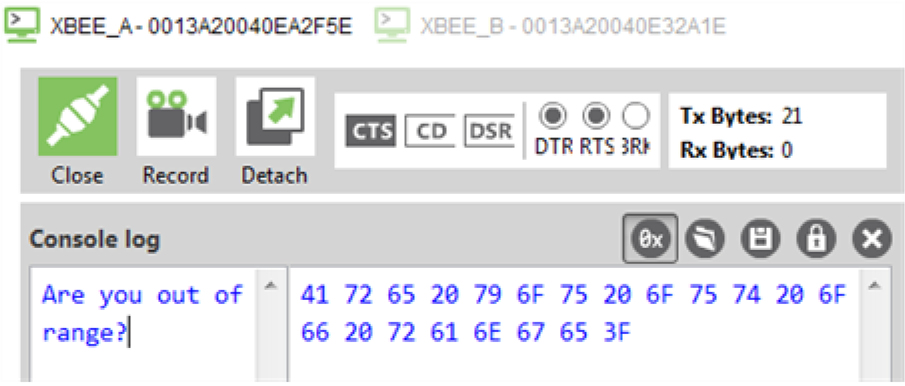
      The approximate indoor range is 500 ft (150 m), and the approximate outdoor range is 2.5 miles (4 km).

c. Make sure the loopback jumper is connected to XBEE_B. See Connect a loopback jumper to an XBee device.

d. In the XBEE_A console, click ⊗ to clear your previous conversation with XBEE_B.

e. Type "Are you out of range?" In the illustration below, the message does not loop back, which means XBEE_B did not receive it and it is out of range of XBEE_A.

f. If the message loops back, move XBEE_B farther away until it no longer loops back.



Tx Bytes: 21
Rx Bytes: 0

```
Are you out of
range?
```

```
41 72 65 20 79 6F 75 20 6F 75 74 20 6F
66 20 72 61 6E 67 65 3F
```

2. Add and configure another XBee module:

   a. Connect another XBee module to your computer.

   b. Click the **Configuration working modes** button ⚙.

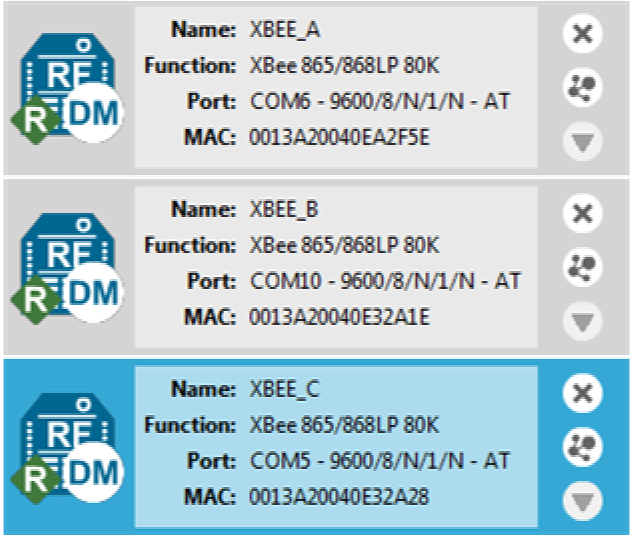
   c. Click the **Add a radio module** button.

   d. In the **Add a radio module** dialog, select the USB Serial Port for this XBee module and click **Finish**.

   e. Configure this XBee module as follows:

   ID: 2015
   NI: XBEE_C

f. Click the **Write radio settings** button .



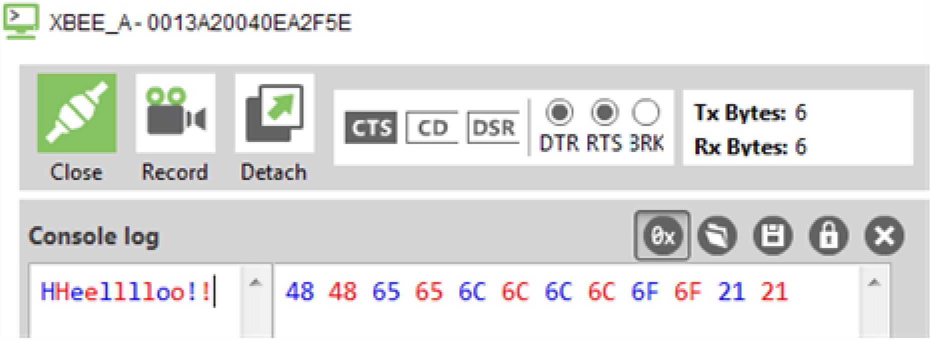
Name: XBEE_A
Function: XBee 865/868LP 80K
Port: COM6 - 9600/8/N/1/N - AT
MAC: 0013A20040EA2F5E

Name: XBEE_B
Function: XBee 865/868LP 80K
Port: COM10 - 9600/8/N/1/N - AT
MAC: 0013A20040E32A1E

Name: XBEE_C
Function: XBee 865/868LP 80K
Port: COM5 - 9600/8/N/1/N - AT
MAC: 0013A20040E32A28

3. Have XBEE_C relay messages between XBEE_A and XBEE_B:
   a. Switch back to the **Consoles working mode** .
   b. Disconnect XBEE_C from your computer and remove it from XCTU.
   c. Connect XBEE_C to a power supply (or laptop or portable battery) and place it between XBEE_A and XBEE_B.
   d. Make sure the loopback jumper is still connected to XBEE_B.
   e. Have XBEE_A send a message to XBEE_B. In the XBEE_A console, type "Hello!"

In the following illustration, the message loops back. XBEE_C relayed your message to XBEE_B, and you successfully established a mesh network.



XBEE_A - 0013A20040EA2F5E

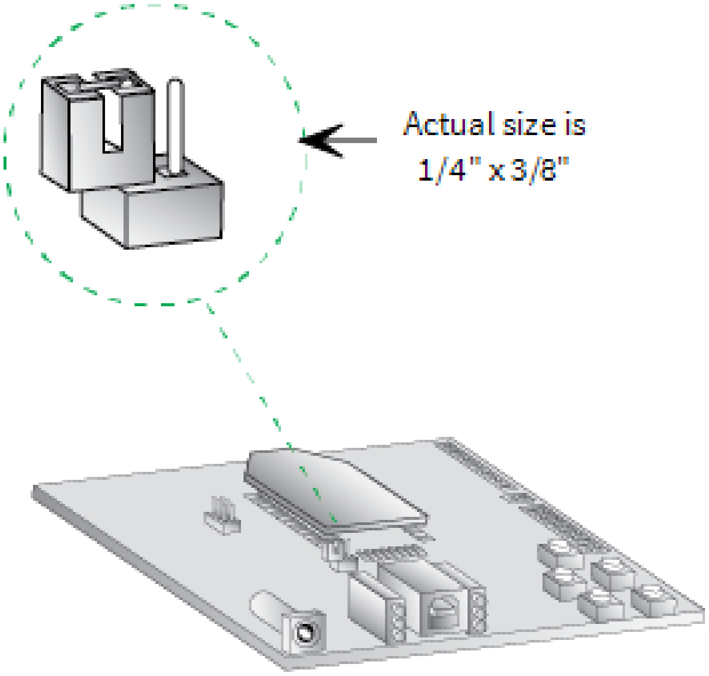
Tx Bytes: 6
Rx Bytes: 6

Console log

```
HHeellllloo!!    48 48 65 65 6C 6C 6C 6C 6F 6F 21 21
```

Tip Use the **Send a single packet** command to send and have an entire message loop back, instead of having individual characters loop back. To do this, click the **Add new packet** button ⊕ to compose your message, and then click **Send selected packet** to send your message.

⚠ Before you perform other tasks, change the loopback jumper on XBEE_B so it no longer bridges the two pins on its development board. It should look like this:



Step 4: Use API mode to talk to XBee modules

This section shows you how to configure an XBee module in API mode, which gives you flexibility, speed, and reliability in your data transmissions.

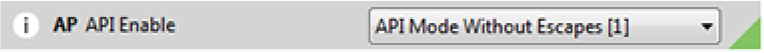
If you get stuck, see Troubleshooting.

For more information on API mode, see Operate in API mode.

Configure a device in API mode

1. Select XBEE_A and click the **Configuration working modes** button .
2. Add this configuration:

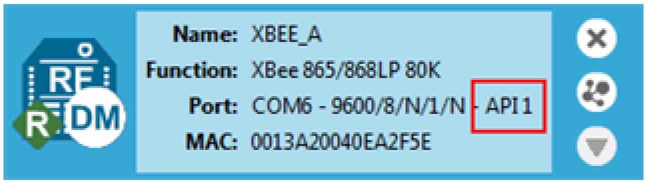
   AP: API Mode 1



AP API Enable: API Mode Without Escapes [1]

3. Click the **Write radio settings** button .

   The Port indicates XBEE_A is in API mode.



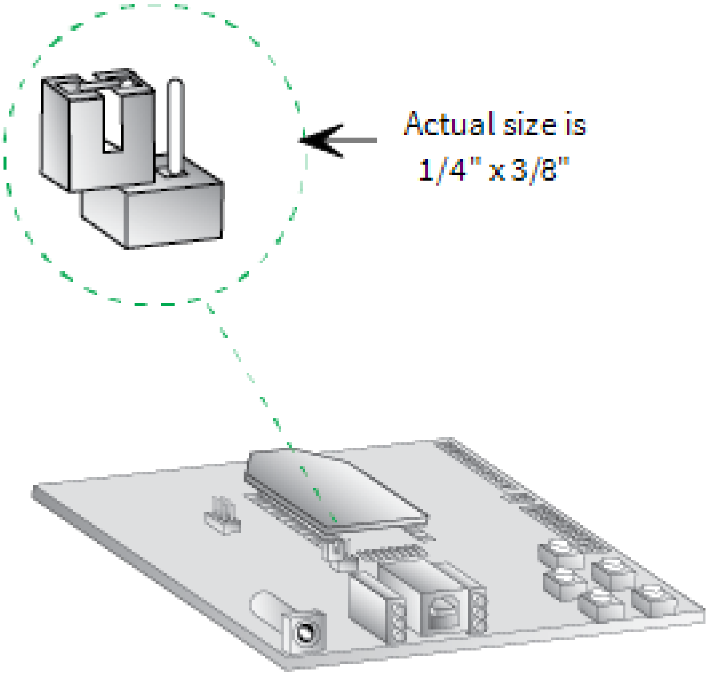
Name: XBEE_A
Function: XBee 865/868LP 80K
Port: COM6 - 9600/8/N/1/N - API 1
MAC: 0013A20040EA2F5E

Send an API Tx frame from an XBee module to another module

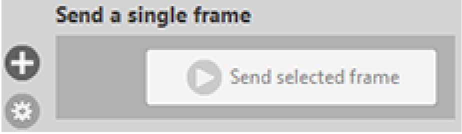
API Tx frames are the instructions that allow one XBee module to send data to another XBee module. In these instructions, XBEE_A uses the API frame type "Transmit Request" to send some text data to

XBEE_B.

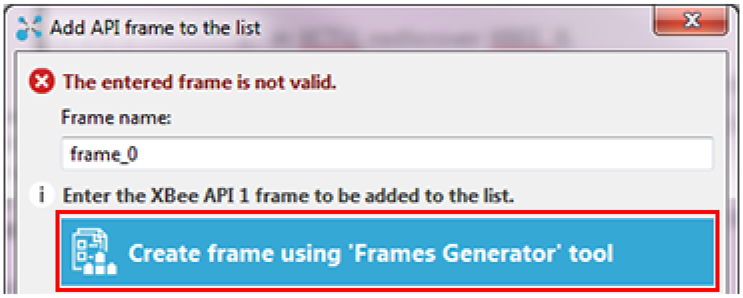
1. Reconnect XBEE_B to your computer.
2. Make sure the loopback jumper on XBEE_B no longer bridges the two pins on its development board.



3. In XCTU, rediscover XBEE_B.
4. Switch XBEE_A and XBEE_B to console mode:
   a. Select **XBEE_A** and click 🖥. Then click 🔌 to open a serial connection.
   b. Select **XBEE_B** and click 🖥. Then click 🔌 to open a serial connection.
5. Select **XBEE_A**.
6. In the **Send a single frame** area, click the **Add new frame to the list** button ⊕.



7. In the **Add API frame to the list** dialog, click the **Create frame using 'Frames Generator' tool** button.



Add API frame to the list

The entered frame is not valid.

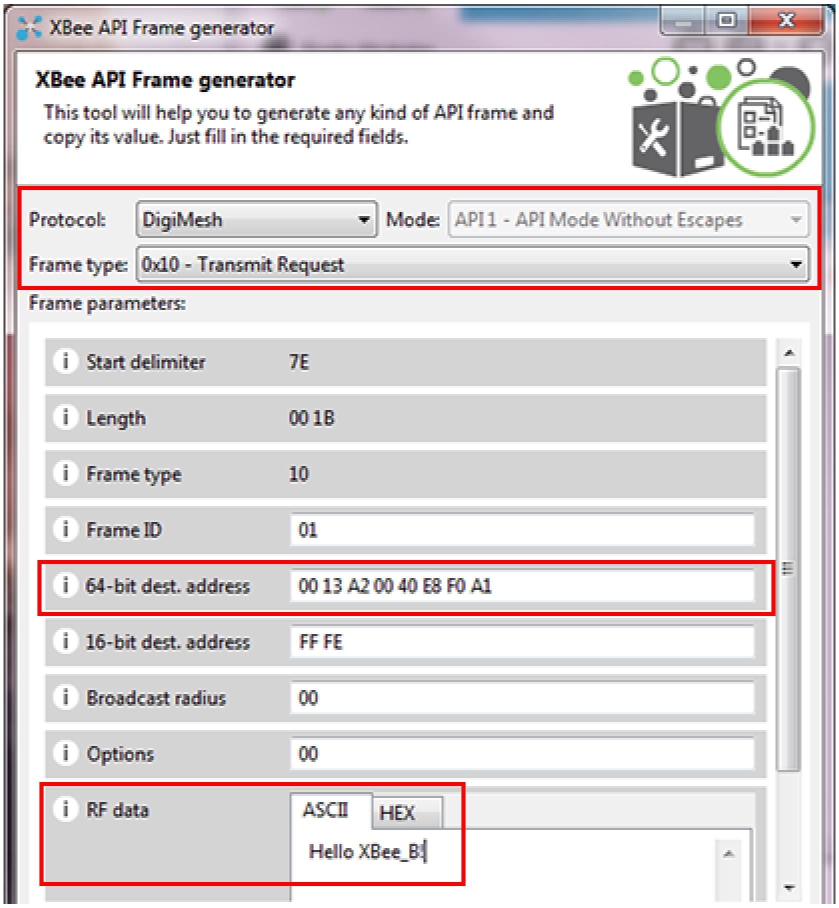
Frame name:

frame_0

Enter the XBee API 1 frame to be added to the list.

8. In the **XBee API Frame generator** dialog, configure the following parameters:

   Protocol: DigiMesh
   Mode: API 1
   Frame type: 0x10 - Transmit Request
   64-bit dest. address: MAC address of XBEE_B
   RF data: Type "Hello XBee_B!" in the ASCII tab



XBee API Frame generator

XBee API Frame generator

This tool will help you to generate any kind of API frame and copy its value. Just fill in the required fields.

Protocol: DigiMesh Mode: API 1 - API Mode Without Escapes

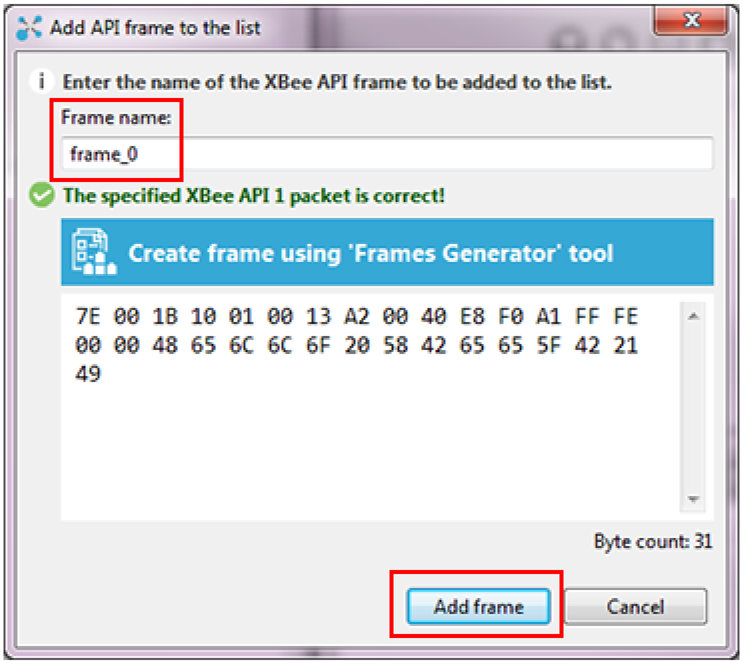
Frame type: 0x10 - Transmit Request

Frame parameters:

Start delimiter	7E
Length	00 1B
Frame type	10
Frame ID	01
64-bit dest. address	00 13 A2 00 40 E8 F0 A1
16-bit dest. address	FF FE
Broadcast radius	00
Options	00
RF data	ASCII HEX Hello XBee_B!

9. Click **OK**.
10. In the **Add API frame to the list** dialog, type a name for your frame.

11. Click **Add frame**.



Add API frame to the list

Enter the name of the XBee API frame to be added to the list.

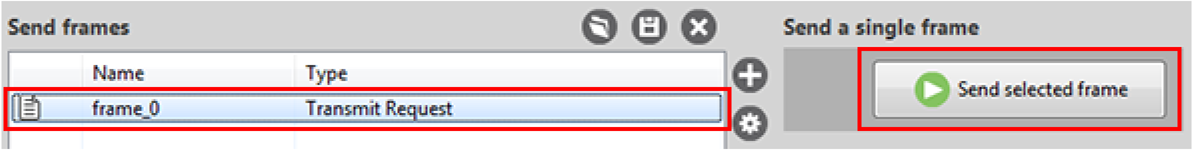
Frame name: frame_0

The specified XBee API 1 packet is correct!

```
7E 00 1B 10 01 00 13 A2 00 40 E8 F0 A1 FF FE
00 00 48 65 6C 6C 6F 20 58 42 65 65 5F 42 21
49
```

Byte count: 31

12. In the **Send frames** area, make sure your frame is selected.
13. In the **Send a single frame** area, click **Send selected frame**.

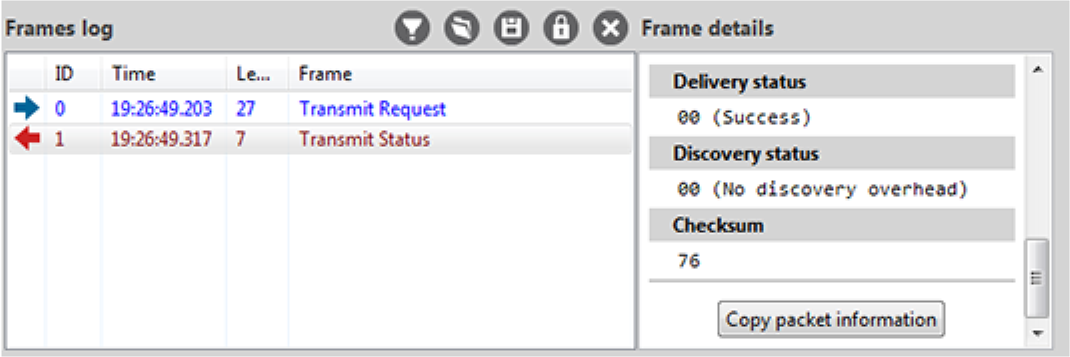


Send frames

Name	Type
frame_0	Transmit Request

14. In the **Frames log** area, select **Transmit Request** and then **Transmit Status** to look at the Frame details for each.

    For example, select **Transmit Status** and scroll down in the **Frame details** area to see that your Delivery status is a success.



Frames log

ID	Time	Le...	Frame
0	19:26:49.203	27	Transmit Request
1	19:26:49.317	7	Transmit Status

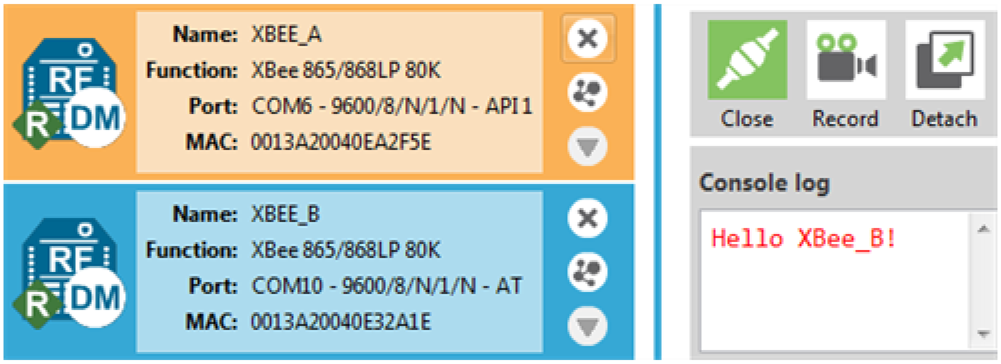
Frame details

Delivery status
00 (Success)

Discovery status
00 (No discovery overhead)

Checksum
76

15. In the **Radio Modules** area, select **XBEE_B**. "Hello XBee_B!" should appear in the Console log.



Name: XBEE_A
Function: XBee 865/868LP 80K
Port: COM6 - 9600/8/N/1/N - API 1
MAC: 0013A20040EA2F5E

Name: XBEE_B
Function: XBee 865/868LP 80K
Port: COM10 - 9600/8/N/1/N - AT
MAC: 0013A20040E32A1E

Console log

```
Hello XBee_B!
```

Do more with your XBee modules

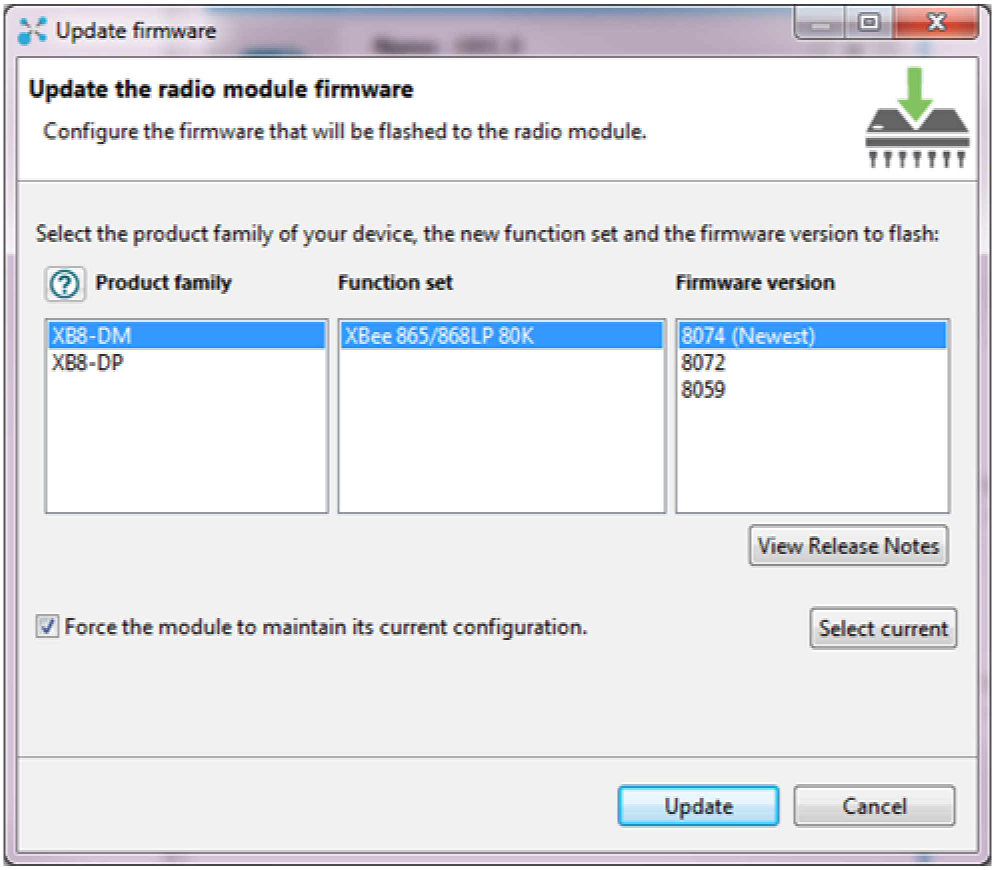
Update the firmware

Radio firmware is the program code stored in the radio module's persistent memory that provides the control program for the device. Use XCTU to update the firmware.

1. Click the **Configuration working modes** button .
2. Add local and remote XBee modules to your computer. See Add devices to XCTU and Configure remote devices.
3. Select a local or remote XBee module from the Radio Modules list.
4. Click the **Update firmware** button .

   The **Update firmware** dialog displays the available and compatible firmware for the selected XBee module.

5. Select the product family of the XBee module, the function set, and the latest firmware version.



Update firmware

Update the radio module firmware

Configure the firmware that will be flashed to the radio module.

Select the product family of your device, the new function set and the firmware version to flash:

Product family	Function set	Firmware version
XB8-DM	XBee 865/868LP 80K	8074 (Newest)
XB8-DP		8072
		8059

☑ Force the module to maintain its current configuration.

6. Click **Update**. A dialog displays update progress.

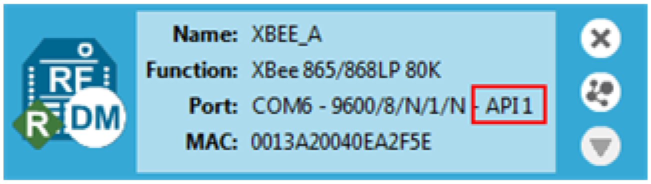
Configure remote devices

You can communicate with remote devices over the air through a corresponding local device. Configure the local device in API mode because remote commands only work in API mode. Configure remote devices in either API or Transparent mode.

These instructions show you how to configure the LT (Associate LED Blink Time) parameter on a remote device.

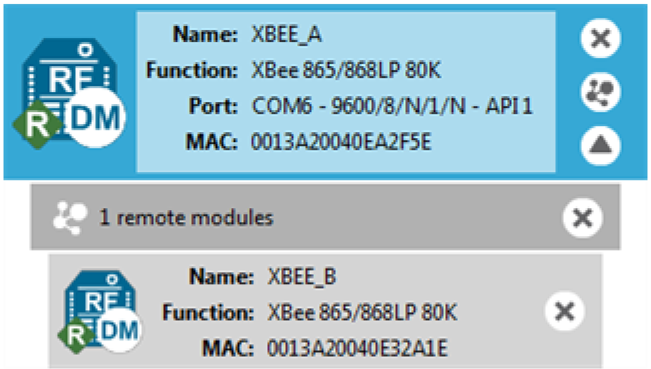
1. Add two XBee devices to XCTU. See Add devices to XCTU.
2. Configure the first device in API mode and name it **XBEE_A**. See Configure a device in API mode.
3. Configure the second device in either API or Transparent mode, and name it **XBEE_B**. See Configure the first two devices in Transparent mode.
4. Disconnect XBEE_B from your computer and remove it from XCTU.
5. Connect XBEE_B to a power supply (or laptop or portable battery).

   The **Radio Modules** area should look something like this.



Name: XBEE_A
Function: XBee 865/868LP 80K
Port: COM6 - 9600/8/N/1/N - API 1
MAC: 0013A20040EA2F5E

6. Select **XBEE_A** and click the **Discover radio nodes in the same network** button .
7. Click **Add selected devices** in the **Discovering remote devices** dialog. The discovered remote device appears below XBEE_A.



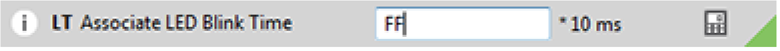
Name: XBEE_A
Function: XBee 865/868LP 80K
Port: COM6 - 9600/8/N/1/N - API 1
MAC: 0013A20040EA2F5E

1 remote modules

Name: XBEE_B
Function: XBee 865/868LP 80K
MAC: 0013A20040E32A1E

8. Select the remote device **XBEE_B**, and configure the following parameter:

   LT: FF (hexidecimal representation for 2550 ms)



LT Associate LED Blink Time FF * 10 ms

9. Click the **Write radio settings** button .

   The remote XBee device now has a different LED blink time.
10. To return to the default LED blink times, change the **LT** parameter back to 0 for XBEE_B.

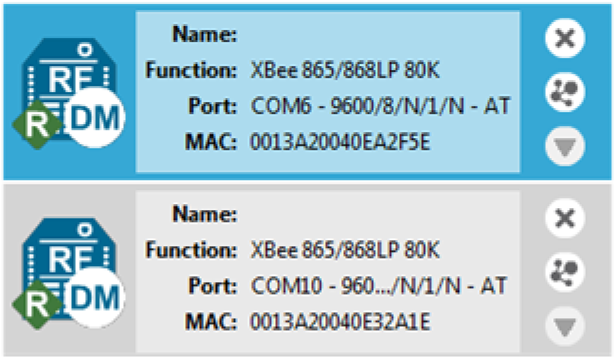
Set up and perform a range test

This section shows you how to set up two XBee modules to perform a range test, which demonstrates the real-world RF range and link quality between two XBee modules in the same network. Performing a range test gives an initial indication of the expected communication performance of the kit components. When deploying an actual network, perform multiple range tests to analyze varying conditions in your application.

Configure the devices for a range test

For devices to communicate with each other, you configure them so they are in the same network. You also set the local device to API mode to obtain all possible data of the remote device.

1. Add two devices to XCTU. See Add devices to XCTU.



Name:
Function: XBee 865/868LP 80K
Port: COM6 - 9600/8/N/1/N - AT
MAC: 0013A20040EA2F5E

Name:
Function: XBee 865/868LP 80K
Port: COM10 - 960.../N/1/N - AT
MAC: 0013A20040E32A1E

2. Select the first device and click the **Load default firmware settings** button.
3. Configure the following parameters:

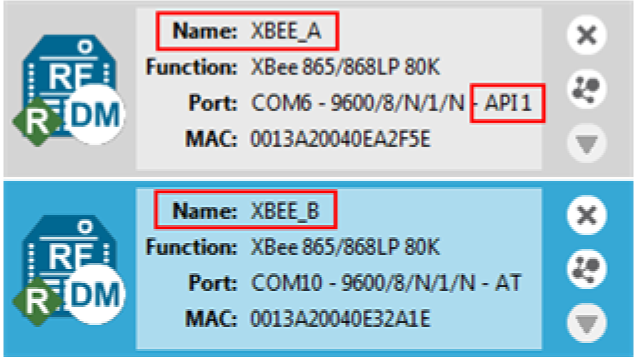
   ID: 2015
   NI: XBEE_A
   AP: API enabled [1]

4. Click the **Write radio settings** button.
5. Select the other device and click.
6. Configure the following parameters:

   ID: 2015
   NI: XBEE_B
   AP: API disabled [0]

7. Click the **Write radio settings** button.

   After you write the radio settings for each device, their names appear in the **Radio Modules** area. The Port indicates XBEE_A is in API mode.



Name: XBEE_A
Function: XBee 865/868LP 80K
Port: COM6 - 9600/8/N/1/N - API 1
MAC: 0013A20040EA2F5E

Name: XBEE_B
Function: XBee 865/868LP 80K
Port: COM10 - 9600/8/N/1/N - AT
MAC: 0013A20040E32A1E

8. Disconnect XBEE_B from the computer and remove it from XCTU.
9. Connect XBEE_B to a power supply (or laptop or portable battery) and move it away from XBEE_A to the desired location for the range test.

   The approximate indoor range is 500 ft (150 m), and the approximate outdoor range is 2.5 miles (4 km).

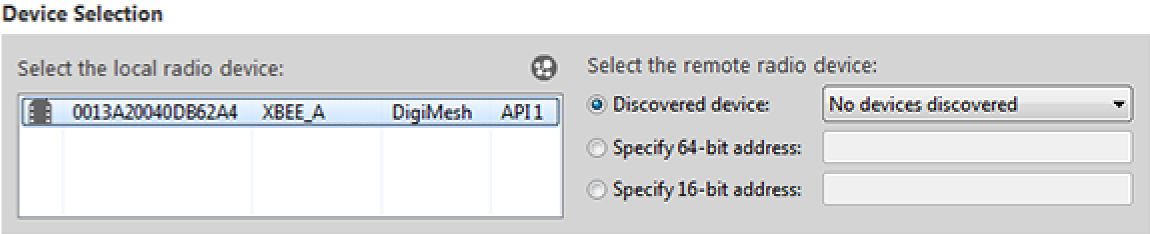
Perform a range test

These instructions show you how to use the loopback cluster (0x12) when performing a range test. The benefit of using this type of range test is you don't have to close the loopback jumper of the remote module and the module can work in any operating mode.

1. In XCTU, open the **Tools** menu and select the **Range Test** option.

   The **Radio Range Test** window opens. Your local device appears on the left side of the **Device Selection** area.

2. Select **XBEE_A** and click the **Discover remote devices** button .



Device Selection

Select the local radio device:

0013A20040DB62A4	XBEE_A	DigiMesh	API 1

Select the remote radio device:

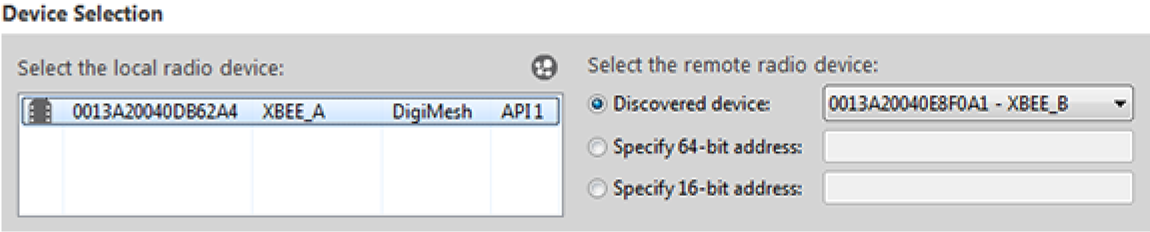
● Discovered device: No devices discovered

○ Specify 64-bit address:

○ Specify 16-bit address:

   The discovery of remote devices starts. When the discovery process finishes, the other device (XBEE_B) appears in the Discovering remote devices dialog.

3. Click **Add selected devices**.
4. Select **XBEE_B** from the **Discovered device** drop-down menu in the **Device Selection** area.



Device Selection

Select the local radio device:

0013A20040DB62A4	XBEE_A	DigiMesh	API 1

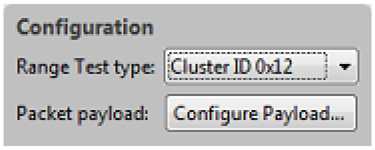
Select the remote radio device:

● Discovered device: 0013A20040E8F0A1 - XBEE_B

○ Specify 64-bit address:

○ Specify 16-bit address:

5. For Range Test type, select **Cluster ID 0x12**.



Configuration

Range Test type: Cluster ID 0x12

Packet payload: Configure Payload...

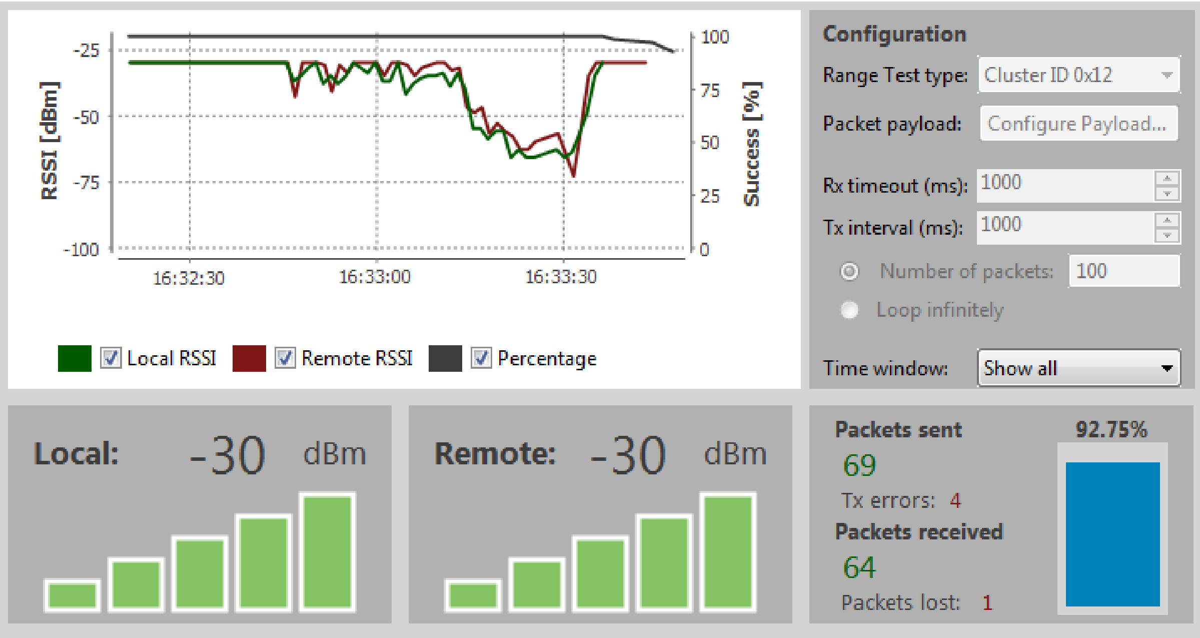
6. Click the **Start Range Test** button Start Range Test.
7. If a notification dialog asks you to close the loopback jumper in the remote device, click **OK**.
8. Test the signal interference by doing one of the following:
   - Place your hands over one of the XBee modules.
   - Block line-of-sight with your body.
   - Place a metal box over an XBee module.
   - Move the remote XBee module to a different room or floor of the building.

   The Received Signal Strength Indicator (RSSI) value will decrease and some packets may even be lost.

9. XCTU represents the retrieved data as follows:
   - **Range Test** charts represent the RSSI values of the local and remote devices during the range test session. The chart also shows the percentage of total packets successfully sent.
   - **Local** and **Remote** bar graphs represent the signal strengths of the local and remote XBee modules. These values are retrieved for the last packet sent/received. RSSI is measured in dBm. A greater negative value in dBm indicates a weaker signal. Therefore, -50 dBm is better than -60 dBm.
   - **Packets sent** and **Packets received** areas show the total number of packets sent, packets received, transmission errors, and packets lost. The percentage bar graph indicates the percentage of packets that are successfully sent and received during a range test session.

   In the following illustration, the percentage of packets successfully sent is 69% and received is 64%. The actual percentage of packets successfully sent or received may be higher.

Range Test



Configuration

Range Test type: Cluster ID 0x12

Packet payload: Configure Payload...

Rx timeout (ms): 1000

Tx interval (ms): 1000

● Number of packets: 100

○ Loop infinitely

Time window: Show all

Local: -30 dBm

Remote: -30 dBm

Packets sent 69

Tx errors: 4

Packets received 64

Packets lost: 1

92.75%

10. Click the **Stop Range Test** button Stop Range Test to stop the process at any time.
11. When you have completed the range test, remove the remote XBee modules from XCTU by clicking the **Remove the list of remote modules** button ⊗.

Configure basic synchronous sleep support

This section shows you how to extend the battery life of an XBee device and demonstrates how a DigiMesh network handles messages when nodes are synchronously sleeping. You will configure one of the devices as a sleep support node and the other two as synchronous cyclic sleep nodes.

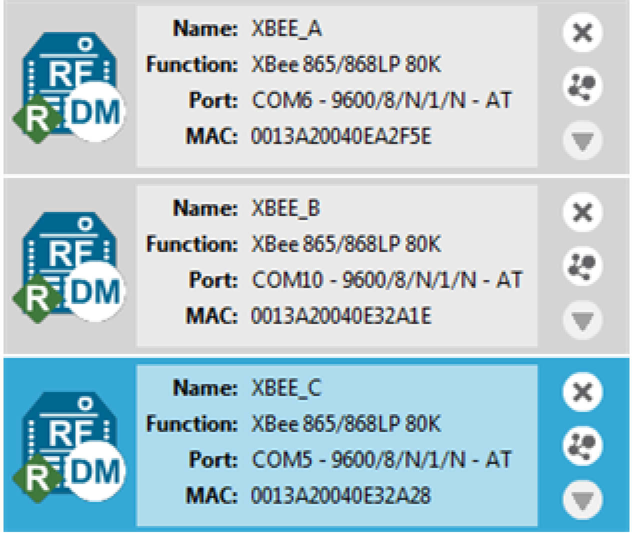
The sleep support XBee device is always awake and can receive serial or over-the-air data at any time, whereas the synchronized sleeping devices cannot send or receive data during their sleep periods. When receivers are asleep, the messages are buffered and forwarded to their destination once they have woken up. In either case, XBee devices can only receive data up to the capacity of the input buffer.

Configure the sleep coordinator for synchronous sleep support

These instructions show you how to configure XBEE_A as the preferred sleep coordinator so it stays awake while the other XBee devices sleep. You then configure XBEE_B and XBEE_C so one of them assumes the role of sleep coordinator when you disconnect XBEE_A. This allows the network to remain in sync with minimal impact on battery life.

Note If you have only two USB cables: After you configure XBEE_B, disconnect it from your computer and remove it from XCTU. Then connect it to a power supply (or laptop or portable battery). Next, use the available USB cable to connect and configure XBEE_C.

1. Add three devices to XCTU. See Add devices to XCTU.
2. For each device, click the **Load default firmware settings** button and then the **Write radio settings** button .
3. Configure the three XBee devices in either Transparent or API mode. This example configures the XBee devices in Transparent mode.



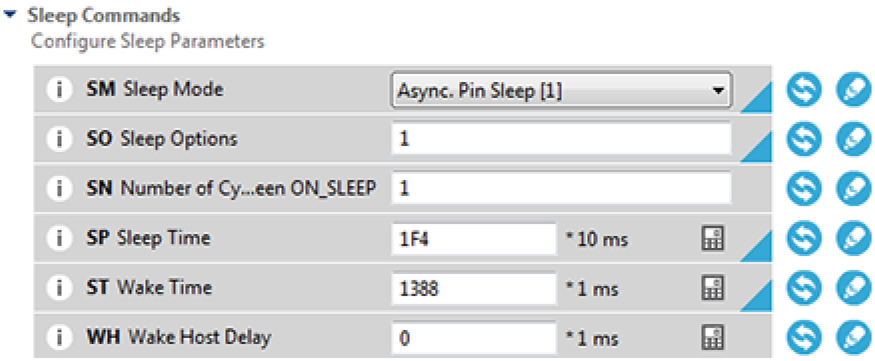
Name: XBEE_A
Function: XBee 865/868LP 80K
Port: COM6 - 9600/8/N/1/N - AT
MAC: 0013A20040EA2F5E

Name: XBEE_B
Function: XBee 865/868LP 80K
Port: COM10 - 9600/8/N/1/N - AT
MAC: 0013A20040E32A1E

Name: XBEE_C
Function: XBee 865/868LP 80K
Port: COM5 - 9600/8/N/1/N - AT
MAC: 0013A20040E32A28

4. Select **XBEE_A** and configure the following parameters:

   SM: 7
   SO: 1 (preferred sleep coordinator)
   SP: 1F4 (hexidecimal) = 500 (decimal) x 10 ms = 5 seconds
   ST: 1388 (hexidecimal) = 5000 (decimal) x 1ms = 5 seconds



Sleep Commands
Configure Sleep Parameters

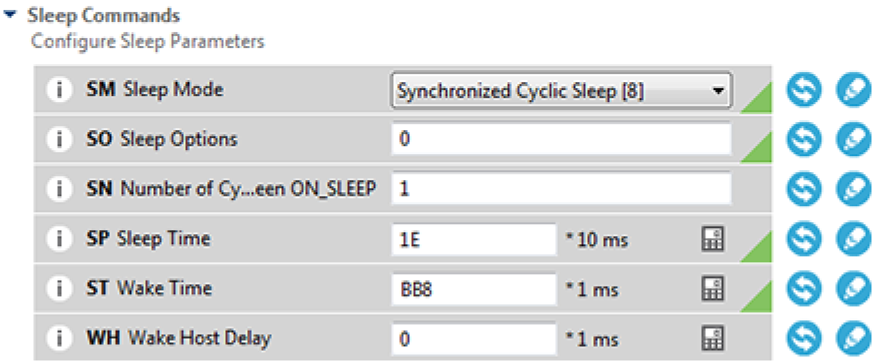
SM Sleep Mode	Async. Pin Sleep [1]	
SO Sleep Options	1	
SN Number of Cy...een ON_SLEEP	1	
SP Sleep Time	1F4	* 10 ms
ST Wake Time	1388	* 1 ms
WH Wake Host Delay	0	* 1 ms

5. Click the **Write radio settings** button ✎.

Note The **SP** (sleep time) and **ST** (wake time) are set to five seconds to make it easy to observe synchronous sleep support. To simulate a sensor system such as water monitoring, you might set **SP** to 30 minutes and **ST** to 10 seconds, depending on the number of devices and amount of data that is transferred.

6. Select **XBEE_B** and configure the following sleep parameters:

   SM: 8
   SO: 0 (allows the XBee module to take over the role of sleep coordinator if the preferred sleep coordinator fails)
   SP: 1E (hexidecimal) = 30 (decimal) x 10 ms = 300 ms
   ST: BB8 (hexidecimal) = 3000 (decimal) x 1 ms = 3 seconds



Sleep Commands
Configure Sleep Parameters

SM Sleep Mode	Synchronized Cyclic Sleep [8]	
SO Sleep Options	0	
SN Number of Cy...een ON_SLEEP	1	
SP Sleep Time	1E	* 10 ms
ST Wake Time	BB8	* 1 ms
WH Wake Host Delay	0	* 1 ms

7. Click the **Write radio settings** button ✎.

8. Configure the sleep parameters for XBEE_C as you did for XBEE_B. Click ✎ when you are done.

Note Once XBEE_B and XBEE_C sync up to the network, their wake and sleep times are controlled by the **OS** and **OW** settings on the sleep support node (XBEE_A). If you want to change the wake and sleep times, change the **SP** and **ST** values for XBEE_A.

9. The LED lights on the three devices appear as follows:

XBee module	Wake period	Sleep period
XBEE_A (sleep coordinator)	Flashing red light	Solid red light
XBEE_B and XBEE_C	Flashing red light	No light

10. Change the role of sleep coordinator:
    a. Disconnect XBEE_A from your computer.
    b. Observe XBEE_B or XBEE_C taking over the role of sleep coordinator by looking at the behavior of the LED lights. It could take three cycles for the new sleep coordinator to take effect.
    c. Re-connect XBEE_A to your computer.
    d. Observe XBEE_A re-assuming the role of sleep coordinator.

Note If a device gets out of sync, it goes through a re-synchronization process.

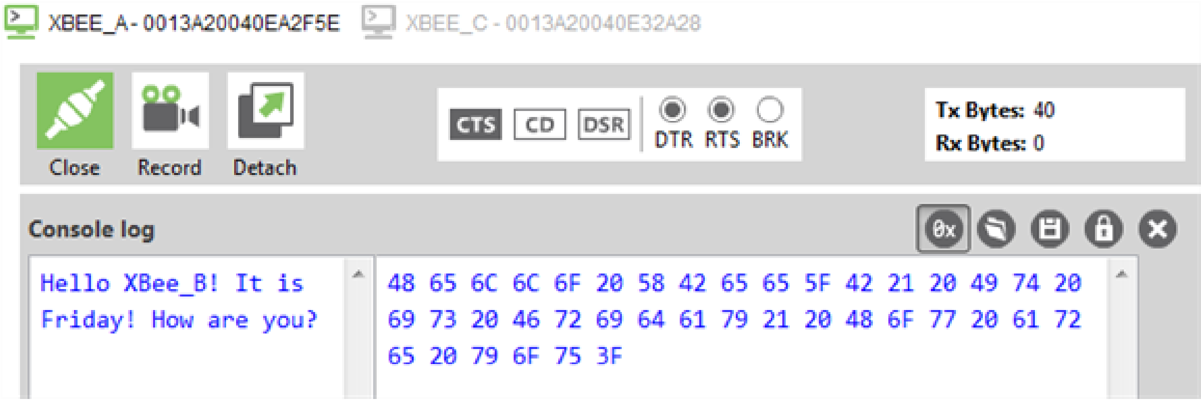
Observe flow control during synchronous sleep support

These instructions demonstrate the importance of observing flow control while XBees are sending and receiving data during synchronous sleep support. Flow control is the process used by a device to inform another device to stop sending data in order to prevent data loss.

1. Click the **Consoles working mode** button .
2. Select **XBEE_A** (the preferred sleep coordinator), and click to open a serial connection.
3. Select **XBEE_B** and click .
4. Click the **Detach view** button to see both consoles at the same time.
5. In the **Console log** area for XBEE_A, type "Hello XBee_B! It is Friday! How are you?"

   Note All XBee DigiMesh modules have a CTS pin (pin 12) that can inform a connected processor when it is permissible to send data to the XBee module. In XCTU, the CTS icon CTS indicates whether an XBee module is awake (the icon is highlighted) or asleep (the icon is not highlighted).

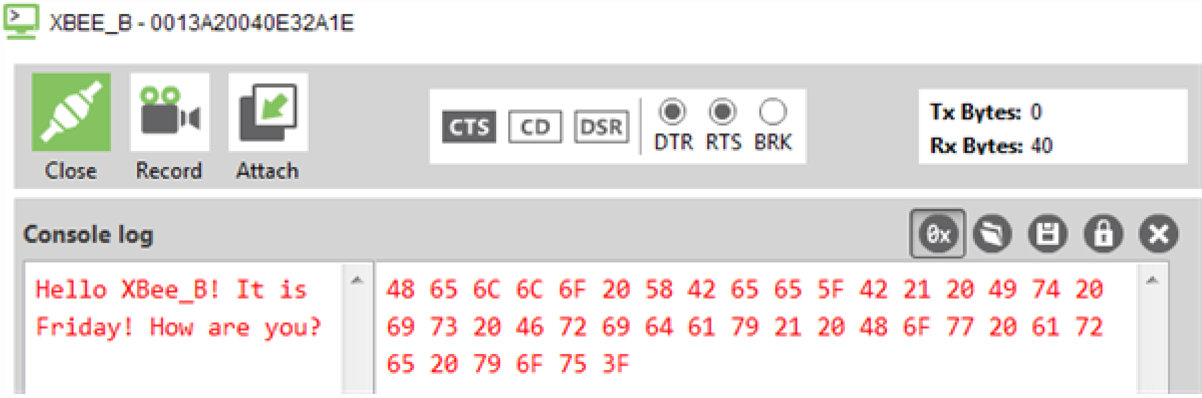
   Since XBEE_A is the sleep coordinator, it transmits its entire message to XBEE_B. The CTS icon for XBEE_A stays on the entire time.



Tx Bytes: 40
Rx Bytes: 0

```
Hello XBee_B! It is
Friday! How are you?
```

```
48 65 6C 6C 6F 20 58 42 65 65 5F 42 21 20 49 74 20
69 73 20 46 72 69 64 61 79 21 20 48 6F 77 20 61 72
65 20 79 6F 75 3F
```



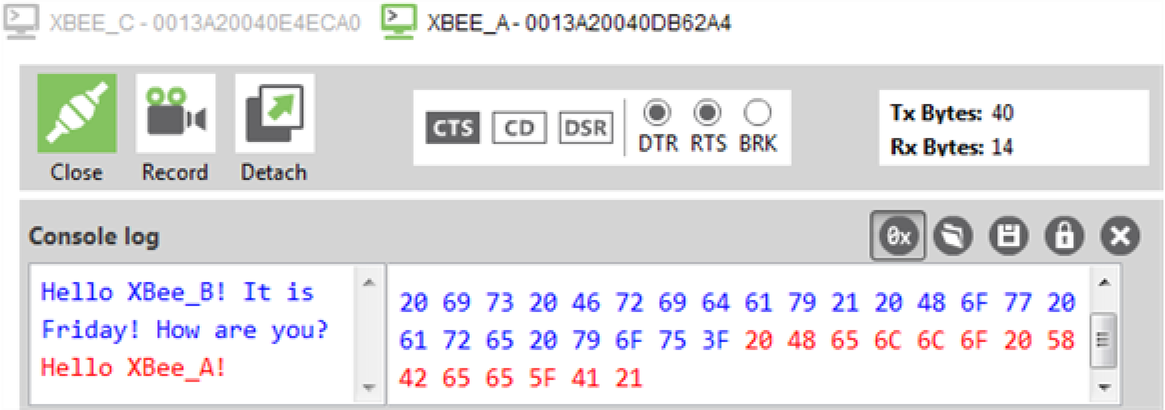
Tx Bytes: 0
Rx Bytes: 40

```
Hello XBee_B! It is
Friday! How are you?
```

```
48 65 6C 6C 6F 20 58 42 65 65 5F 42 21 20 49 74 20
69 73 20 46 72 69 64 61 79 21 20 48 6F 77 20 61 72
65 20 79 6F 75 3F
```

6. In the **Console log** area for XBEE_B, type "Hello XBee_A! It is Friday! How are you?"

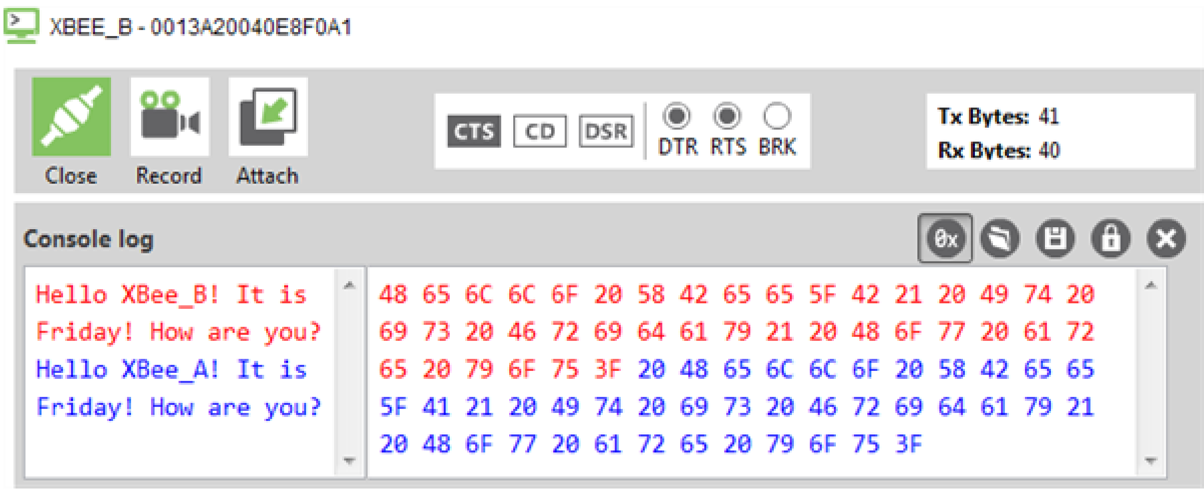
   Since XBEE_B is a synchronized sleeping module, it only transmits the part of the message that is typed while it is awake. In the illustration below, it was only able to transmit "Hello XBee_A!" The CTS icon for XBEE_B turned off after this part of the message was typed.



Tx Bytes: 40
Rx Bytes: 14

```
Hello XBee_B! It is
Friday! How are you?
Hello XBee_A!
```

```
20 69 73 20 46 72 69 64 61 79 21 20 48 6F 77 20
61 72 65 20 79 6F 75 3F 20 48 65 6C 6C 6F 20 58
42 65 65 5F 41 21
```


XBEE_B - 0013A20040E8F0A1

Tx Bytes: 41
Rx Bytes: 40

```
Hello XBee_B! It is      48 65 6C 6C 6F 20 58 42 65 65 5F 42 21 20 49 74 20
Friday! How are you?     69 73 20 46 72 69 64 61 79 21 20 48 6F 77 20 61 72
Hello XBee_A! It is      65 20 79 6F 75 3F 20 48 65 6C 6C 6F 20 58 42 65 65
Friday! How are you?     5F 41 21 20 49 74 20 69 73 20 46 72 69 64 61 79 21
                         20 48 6F 77 20 61 72 65 20 79 6F 75 3F
```

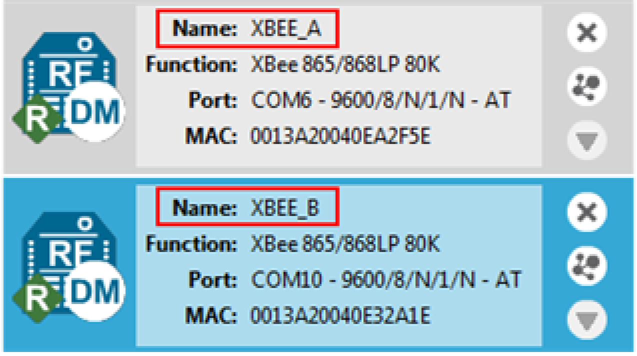
7. To disconnect, click the **Close serial connection** button for each console.

Set up basic encryption for an XBee network

The information transmitted in an XBee network sometimes needs to be protected. For example, an XBee network transferring financial information must be carefully protected against external agents. These instructions show you how to configure XBee 868LP RF Modules for secure communication via encryption keys.

Note You can use encryption for devices that have been configured for either Transparent or API mode.

1. Add two XBee modules to XCTU. See Add devices to XCTU.
2. Configure the XBee modules so they can talk to each other. See Configure the first two devices in Transparent mode.
3. Name your two XBee modules **XBee_A** and **XBee_B**.

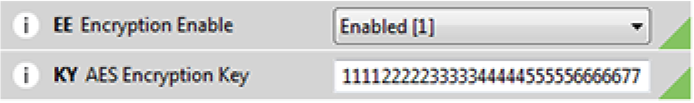


Name: XBEE_A
Function: XBee 865/868LP 80K
Port: COM6 - 9600/8/N/1/N - AT
MAC: 0013A20040EA2F5E

Name: XBEE_B
Function: XBee 865/868LP 80K
Port: COM10 - 9600/8/N/1/N - AT
MAC: 0013A20040E32A1E

4. Select **XBee_A** and configure the following parameters:

   EE: Set the **AES Encryption Enable** parameter to 1.
   KY: Set the **AES Encryption Key** parameter to a 32 hexadecimal character string. Example: 11111222223333344444555556666677



EE Encryption Enable: Enabled [1]

KY AES Encryption Key: 1111222223333344444555556666677

5. Click the **Write radio settings** button ✎ .
6. Configure the parameters for XBEE_B as you did for XBEE_A, and then click ✎ .
7. Send a secure message between XBee_A and XBee_B. See Send messages through XCTU.

Note If you add more devices, give them the same encryption key so they can communicate with the other XBee devices.

8. To return to the encryption disabled setting, change the **EE** parameter back to **0** for XBEE_A and XBEE_B.

Learn more about XBee module features

For more information about XBee 868LP RF modules, see the *XBee 868LP RF Modules User Guide*. You can find this guide on the Digi Support site.

Unicast versus broadcast transmissions

An XBee module can communicate with multiple devices or with just one device:

- Broadcast transmissions are sent to many or all XBee modules in the network.
- Unicast transmissions route wireless data from one XBee module to another specific XBee module.

Broadcast transmission

A broadcast transmission transmits the same data to all nodes on the network. These transmissions are propagated throughout the entire network so that all possible nodes receive the transmission.

An example of broadcast communication is a television station.

Unicast transmission

A unicast transmissionsends messages to a single node on the network that is identified by a unique 64-bit address. The destination XBee module could be an immediate neighbor of the sender, or be several hops away.

An example of a unicast communication is a telephone call between two people.

For more information, see Data transmission and routing.

Analog inputs and digital inputs and outputs

All XBee modules have a set of pins that can be used to connect sensors or actuators and configure them for specific behavior. Each XBee module has the capability to directly gather sensor data and transmit it without the use of an external microcontroller.

With these pins you can, for example, turn on a light by sending information to an XBee module connected to an actuator, or measure the outside temperature by obtaining data from a temperature sensor attached to your XBee module.

Sleep modes

Putting XBee devices into a temporary sleep state preserves battery life when using wireless networks. DigiMesh devices support five sleep modes that are classified as synchronous or asynchronous. For more information about using sleep modes, see Sleep modes.

Note Asynchronous sleep modes should not be used in a synchronous sleeping network, and vice versa.

Transparent and API operating modes

The firmware operates in several different modes. Two top-level modes establish how the device communicates with other devices through its serial interface: Transparent operating mode and API operating mode.

Transparent operating mode

Devices operate in this mode by default. We also call this mode “AT operating mode.” The device acts as a serial line replacement when it is in Transparent operating mode. The device queues all of UART data it receives through the DIN pin for RF transmission. When a device receives RF data, it sends the data out through the DOUT pin. You can set the configuration parameters using the AT Command interface.

API operating mode

API operating mode is an alternative to Transparent mode. API mode is a frame-based protocol that allows you to direct data on a packet basis. It can be particularly useful in large networks where you need to control the route a data packet takes or when you need to know which node a data packet is from. The device communicates UART data in packets, also known as API frames. This mode allows for structured communications with serial devices. It is helpful in managing larger networks and is more appropriate for performing tasks such as collecting data from multiple locations or controlling multiple devices remotely.

There are two types of API operating modes: one with escaped characters and another without escaped characters.

- Without escaped characters. This mode eliminates escaping character sequences. This makes it simpler to create code and libraries, but runs a minor risk of lost frames or errors due to the possibility that payload data can be confused with frame structure. We do not recommend this mode for noisy radio environments and where payload data may include special characters (specifically 0x7E, 0x7D, 0x11, and 0x13).
- With escaped characters. This mode escapes characters in an API frame in order to improve the reliability of the RF transmission, especially in noisy environments. API escaped operating mode (AP = 2) works similarly to API mode. The only difference is that when working in API escaped mode, the software must escape any payload bytes that match API frame specific data, such as the start-of-frame byte (0x7E).

Troubleshooting

If you get stuck while performing any of the tasks in this guide, try one of these troubleshooting tips.

Cannot install device driver

Device driver software was not successfully installed.

Condition

Sometimes when you connect an XBee module to your computer, the operating system does not install the driver.

Solution

Try the following, in order. If one of the steps resolves the issue, you're done.

1. Remove and re-insert the XBee module into your computer.
2. If the OS is still unable to install the driver, remove and re-insert the XBee module into another USB port.
3. Manually install the USB drivers. See Optional: Manually install USB drivers.

Use LEDs to identify XBee modules

You want to force LEDs to blink so you can easily locate an XBee 868LP RF Module.

Resolution

To locate an XBee 868LP RF Module using LEDs:

1. In XCTU, select one of the devices and click the **Read radio settings** button .
2. Observe which device has the Rx and Tx LED lights blinking green and yellow on its development board.

No remote devices to select for a range test

If there are no remote devices to select in the Radio Range Test dialog, try one of the following resolutions.

Check cables

The USB cables should be firmly and fully attached to both the computer and the development board. When attached correctly, the association LED on the adapter is illuminated.

Check that the device is fully seated in the development board

When the device is correctly installed, it is pushed fully into the board and no air or metal is visible between the plastic of the adapter socket and the XBee 868LP RF Module headers. Also, check that all ten pins on each side of the device are in a matching hole in the socket.

Check the XBee device orientation

The angled "nose" of the XBee 868LP RF Module should match the lines on the silk screening of the board and point away from the USB socket on the development board.

Check that the devices are in the same network

Check that the following parameters have the same value for all devices on the network:

XBee module development kit	Parameters
XBee/XBee-PRO DigiMesh 2.4	**ID** (Network ID) and **CH** (Operating Channel)
XBee S2C DigiMesh 2.4	**ID** (Network ID) and **CH** (Operating Channel)
XBee 868LP	**ID** (Network ID), **HP** (Preamble ID), and **CM** (Channel Mask)

Restore default settings

If the devices are properly connected and in the same network, restore default settings and configure them again.

Port in use

Message: "The port is already in use by other applications."

Condition

The serial port where the local XBee 868LP RF Module is connected can only be in use by one application.

Solution

Make sure the connection with the XBee 868LP RF Module in the XCTU console is closed and there are no other applications using the port.

XCTU cannot discover devices

If XCTU does not discover an XBee device or does not display any serial ports, try the following resolutions.

Check the configuration of your USB serial converter

1. On the **Start** menu, click **Computer** > **System Properties** > **Device Manager**.
2. Under Serial Bus controllers, double-click the first USB Serial Converter to open the USB Serial Converter dialog.
3. Click the **Advanced** tab, make sure **Load VCP** is selected, and click **OK**.
4. Repeat steps 2 and 3 for each USB Serial Converter listed in the Device Manager.

Check cables

Double-check all cables. The USB cable should be firmly and fully attached to both the computer and the XBee development board. When attached correctly, the association LED on the adapter will be lit.

Check that the XBee module is fully seated in the XBee development board

When the XBee module is correctly installed, it should be pushed fully into the board and no air or metal should be visible between the plastic of the adapter socket and the XBee module headers. Also, double-check that all ten pins on each side of the XBee module made it into a matching hole in the socket.

Check the XBee module orientation

The angled "nose" of the XBee module should match the lines on the silk screening of the board and point away from the USB socket on the XBee development board.

Check driver installation

Drivers are installed the first time the XBee development board is plugged in. If this process is not complete or has failed, try the following steps:

1. Remove and re-insert the board into your computer. This may cause driver installation to re-occur.
2. Remove and re-insert the board into another USB port.
3. (Windows) Open Computer management, find the failing device in the Device Manager section and remove it.
4. Download the appropriate driver. You can download drivers for all major operating systems from FTDI for manual installation.

Check if the modules are sleeping

The On/Sleep LED of the XBee development board indicates if the XBee module is awake (LED on) or asleep (LED off). When an XBee module is sleeping, XCTU cannot discover it, so press the Commissioning button to wake it up for 30 seconds.

XCTU cannot discover remote devices

XCTU does not discover remote XBee 868LP RF Module.

Potential cause

The devices do not have the appropriate values for the following parameters:

XBee module development kit	Parameters
XBee 868LP	**ID** (Network ID), **HP** (Preamble ID), and **CM** (Channel Mask)

Resolution

1. Ensure that all devices on your network have the same value for each of the parameters listed in the table.
2. If this does not resolve the issue, try setting your devices back to their default settings. Select each XBee device and click the **Load default firmware settings** button 🏭.

XCTU cannot discover remote devices for a range test

When setting up a range test in the Radio Range Test dialog, you receive the message "There are not remote devices discovered for the selected local device."

Condition

In the Radio Range Test dialog, the local radio device you selected has not yet discovered any remote devices.

Solution

In the Device Selection area in the Radio Range Test dialog, click the **Discover remote devices** button and XCTU will discover devices on the local device's network.

XCTU installation error

An error is reported when installing XCTU.

Condition

XCTU requires Administrator permissions.

Solution

Check that you have Administrator access on the computer where you are installing XCTU. On Windows systems, a User Account Control dialog may appear when you install XCTU or try to run the XCTU program. You must answer yes when prompted to allow the program to make changes to your computer, or XCTU will not work correctly. Note that you may also need to talk to your network manager to gain permission to install or run applications as administrator.

Configure the XBee 868LP RF Module

Software libraries

One way to communicate with the XBee 868LP RF Module is by using a software library. The libraries available for use with the XBee 868LP RF Module include:

- XBee Java library
- XBee Python library

The XBee Java Library is a Java API. The package includes the XBee library, its source code and a collection of samples that help you develop Java applications to communicate with your XBee devices.

The XBee Python Library is a Python API that dramatically reduces the time to market of XBee projects developed in Python and facilitates the development of these types of applications, making it an easy process.

XBee Network Assistant

The XBee Network Assistant is an application designed to inspect and manage RF networks created by Digi XBee devices. Features include:

- Join and inspect any nearby XBee network to get detailed information about all the nodes it contains.
- Update the configuration of all the nodes of the network, specific groups, or single devices based on configuration profiles.
- Geo-locate your network devices or place them in custom maps and get information about the connections between them.
- Export the network you are inspecting and import it later to continue working or work offline.
- Use automatic application updates to keep you up to date with the latest version of the tool.

See the *XBee Network Assistant User Guide* for more information.

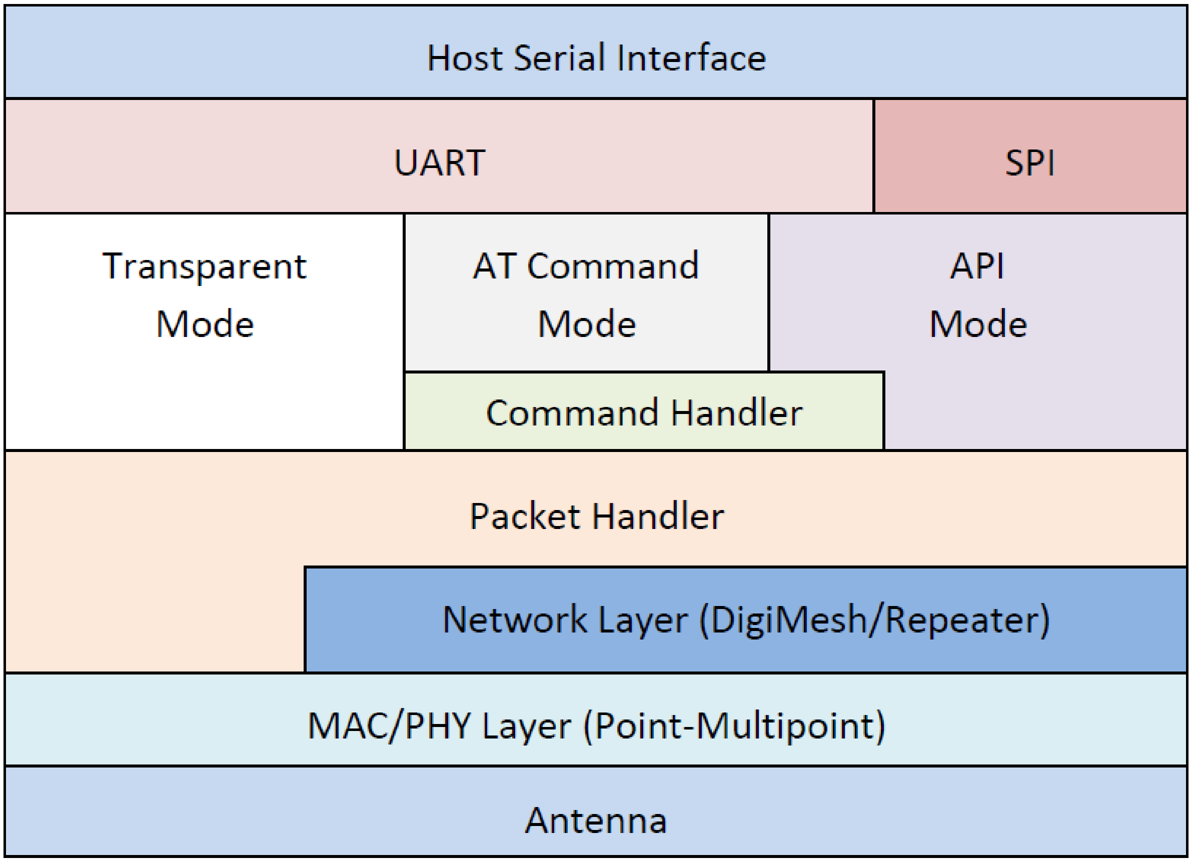
To install the XBee Network Assistant:

1. Navigate to digi.com/xbeenetworkassistant.
2. Click **General Diagnostics, Utilities and MIBs**.
3. Click the **XBee Network Assistant - Windows x86** link.
4. When the file finishes downloading, run the executable file and follow the steps in the XBee Network Assistant Setup Wizard.

Operation

Operation

The XBee 868LP RF Module uses a multi-layered firmware base to order the flow of data, dependent on the hardware and software configuration you choose. The following configuration block diagram shows the host serial interface as the physical starting point and the antenna as the physical endpoint for the transferred data. A block must be able to touch another block above or below it for the two interfaces to interact. For example, if the device uses SPI mode, Transparent mode is not available as shown in the following image:



The command handler code processes commands from AT Command Mode or API Mode; see AT commands. The command handler also processes commands from remote devices; see API frame exchanges.

Listen Before Talk and Automatic Frequency Agility

This device implements Listen Before Talk (LBT) and Automatic Frequency Agility (AFA). The advantage of LBT with AFA is that the device bypasses the Duty Cycle requirement imposed by European standards. LBT+AFA requires that you use at least two frequencies for transmission.

This feature provides a level of fairness to the devices in a given area. Before this device transmits, it senses a channel to determine if there is activity by taking an RSSI measurement for 5 ms. If the measurement is below the threshold, the device transmits on that channel. If there is activity, that channel is not used, and the device listens for at least 5 ms to allow transmissions to be received.

After the device transmits on a channel, it will not transmit on that channel again until the minimum TX off time has been met, which is greater than 100 ms. It is useful to have many channels in your channel mask, so transmissions are less likely to be delayed.

European requirements also state that only 100 seconds of transmission may occur over the period of an hour on 200 kHz of spectrum. This method simplifies and optimizes the calculations of spectrum use over the period of one hour. The standard states that the more channels you have, the more transmission time you have in a one hour period. Calculate the effective duty cycle based on the number of available channels enabled as follows:

Effective Duty Cycle = (number of channels * 100)/3600.

For example, if you enabled two channels you would have an effective duty cycle of 5.6%.

The XBee 868LP RF Module uses a sliding bucket algorithm to calculate usage over the period of 1 hour for each channel. Each bucket accumulates for 6 minutes.

This device has a maximum of 30 AFA channels to choose from, and channels can be excluded by setting the channel mask (**CM**) to reduce them. Since not all countries allow for all of these channels, the set may be dramatically smaller for some countries. For a complete list, refer to www.digi.com.

Single frequency mode band mode

When you set the channel mask to 0x20000000, the device is in a single frequency mode, and the frequency is 869.85 MHz. In this mode:

- LBT+AFA mode is disabled.
- The device assumes no duty cycle requirement (or 100% duty cycle).
- The **PL** setting must be set to 5 mW to comply with the single frequency mode requirements.

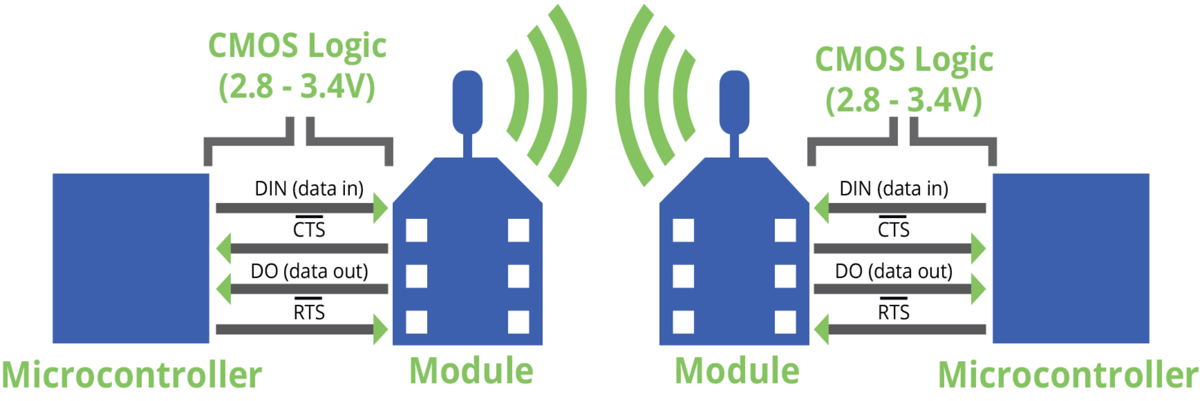
Serial communications

RF Modules interface to a host device through a serial port. Using its serial port, the device communicates with any of the following:

- Logic and voltage compatible UART
- Level translator to any serial device (for example, through an RS-232 or USB interface board)

UART data flow

Devices that have a UART interface connect directly to the pins of the XBee 868LP RF Module as shown in the following figure. The figure shows system data flow in a UART-interfaced environment. Low-asserted signals have a horizontal line over the signal name.

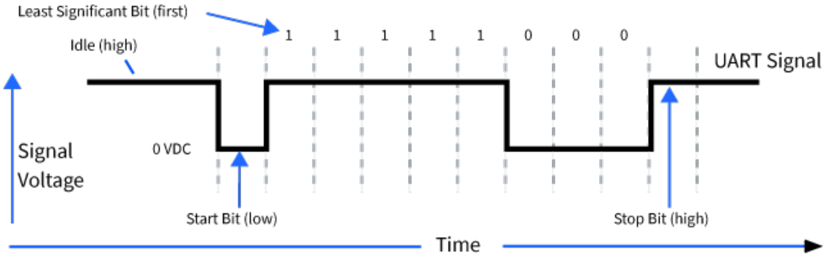


Serial data

A device sends data to the XBee 868LP RF Module's UART through pin 4 DIN as an asynchronous serial signal. When the device is not transmitting data, the signals should idle high.

For serial communication to occur, you must configure the UART of both devices (the microcontroller and the XBee 868LP RF Module) with compatible settings for the baud rate, parity, start bits, stop bits, and data bits.

Each data byte consists of a start bit (low), 8 data bits (least significant bit first) and a stop bit (high). The following diagram illustrates the serial bit pattern of data passing through the device. The diagram shows UART data packet 0x1F (decimal number 31) as transmitted through the device.



You can configure the UART baud rate, parity, and stop bits settings on the device with the **BD**, **NB**, and **SB** commands respectively. For more information, see Serial interfacing commands.

SPI communications

The XBee 868LP RF Module supports SPI communications in slave mode. Slave mode receives the clock signal and data from the master and returns data to the master. The following table shows the signals that the SPI port uses on the device.

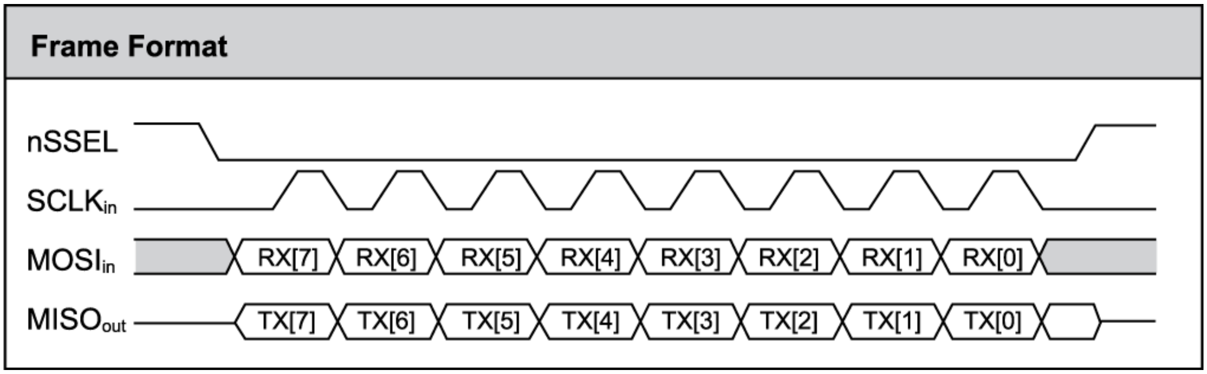
Signal	Function
SPI_MOSI (Master Out, Slave In)	Inputs serial data from the master
SPI_MISO (Master In, Slave Out)	Outputs serial data to the master
SPI_SCLK (Serial Clock)	Clocks data transfers on MOSI and MISO
SPI_$\overline{\text{SSEL}}$ (Slave Select)	Enables serial communication with the slave
SPI_$\overline{\text{ATTN}}$ (Attention)	Alerts the master that slave has data queued to send. The XBee 868LP RF Module asserts this pin as soon as data is available to send to the SPI master and it remains asserted until the SPI master has clocked out all available data.

In this mode:

- SPI clock rates up to 3.5 MHz are possible.
- Data is most significant bit (MSB) first.

- Frame Format mode 0 is used. This means CPOL= 0 (idle clock is low) and CPHA = 0 (data is sampled on the clock's leading edge).
- The SPI port only supports API Mode (**AP** = **1**).

The following diagram shows the frame format mode 0 for SPI communications.



SPI operation

This section specifies how SPI is implemented on the device, what the SPI signals are, and how full duplex operations work.

SPI implementation

The XBee 868LP RF Module operates as a SPI slave only. This means an external master provides the clock and decides when to send data. The XBee 868LP RF Module supports an external clock rate of up to 3.5 Mb/s.

The device transmits and receives data with the most significant bit first using SPI mode 0. This means the CPOL and CPHA are both 0. We chose Mode 0 because it is the typical default for most microcontrollers and simplifies configuring the master.

SPI signals

The specification for SPI includes the four signals: SPI_MISO, SPI_MOSI, SPI_CLK, and SPI_SSEL. Using only these four signals, the master cannot know when the slave needs to send and the SPI slave cannot transmit unless enabled by the master. For this reason, the SPI_ATTN signal is available. This allows the device to alert the SPI master that it has data to send. In turn, the SPI master asserts SPI_SSEL and starts SPI_CLK unless these signals are already asserted and active respectively. This allows the XBee 868LP RF Module to send data to the master.

The following table names the SPI signals and specifies their pinouts. It also describes the operation of each pin.

Signal name	Pin number	Applicable AT command	Description
SPI_MISO (Master In, Slave out)	17	**P5**	When SPI_SSEL is asserted (low) and SPI_CLK is active, the device outputs the data on this line at the SPI_CLK rate. When SPI_SSEL is de-asserted (high), this output should be tri-stated such that another slave device can drive the

Signal name	Pin number	Applicable AT command	Description
			line.
SPI_MOSI (Master out, Slave in)	16	**P6**	The SPI master outputs data on this line at the SPI_CLK rate after it selects the desired slave. When you configure the device for SPI operations, this pin is an input.
SPI_$\overline{\text{SSEL}}$ (Slave Select) (Master out, Slave in)	15	**P7**	The SPI master outputs a low signal on this line to select the desired slave. When you configure for SPI operations, this pin is an input.
SPI_CLK (Clock) (Master out, Slave in)	14	**P8**	The SPI master outputs a clock on this pin, and the rate must not exceed the maximum allowed, 3.5 Mb/s. When you configure the device for SPI operations, this pin is an input.
SPI_$\overline{\text{ATTN}}$ (Attention) (Master in, Slave out)	12	**P9**	The device asserts this pin low when it has data to send to the SPI master. When you configure for SPI operations, it is an output (not tri-stated).

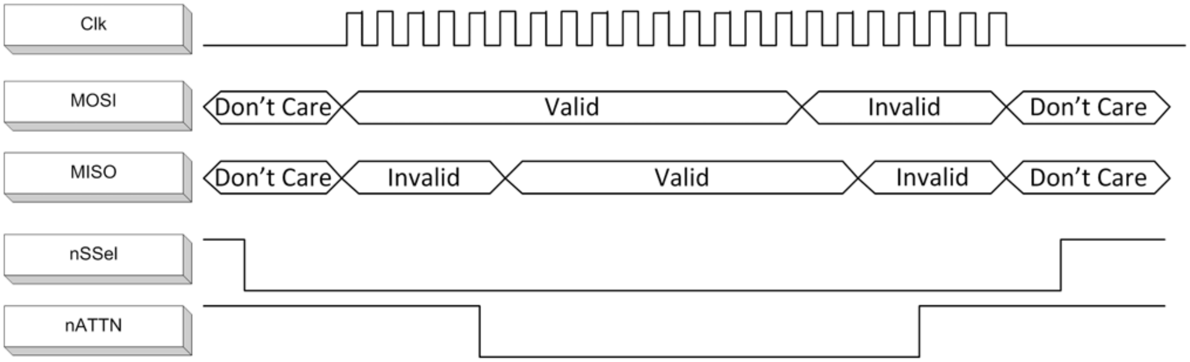
Note By default, the inputs have pull-up resistors enabled. See PR (Pull-up/Down Resistor Enable) to disable the pull-up resistors. When the SPI pins are not connected but the pins are configured for SPI operation, the pull-ups are required for proper UART operation.

Full duplex operation

SPI on the XBee 868LP RF Module requires that you use API mode (without escaping) to packetize data. By design, SPI is a full duplex protocol even when data is only available in one direction. This means that when a device receives data, it also transmits and that data is normally invalid. Likewise, when the device transmits data, invalid data is probably received. To determine whether or not received data is invalid, we packetize the data with API packets.

SPI allows for valid data from the slave to begin before, at the same time, or after valid data begins from the master. When the master is sending data to the slave and the slave has valid data to send in the middle of receiving data from the master, this allows a true full duplex operation where data is valid in both directions for a period of time. Not only must the master and the slave both be able to keep up with the full duplex operation, but both sides must honor the protocol as specified.

The following diagram illustrates the SPI interface while valid data is being sent in both directions.



Low power operation

Sleep modes generally work the same on SPI as they do on UART. However, due to the addition of SPI mode, there is an option of another sleep pin, as described below.

By default, Digi configures DIO8 (SLEEP_REQUEST) as a peripheral and during pin sleep it wakes the device and puts it to sleep. This applies to both the UART and SPI serial interfaces.

If SLEEP_REQUEST is not configured as a peripheral and SPI_SSEL is configured as a peripheral, then pin sleep is controlled by SPI_SSEL rather than by SLEEP_REQUEST. Asserting SPI_SSEL (pin 15) by driving it low either wakes the device or keeps it awake. Negating SPI_SSEL by driving it high puts the device to sleep.

Using SPI_SSEL to control sleep and to indicate that the SPI master has selected a particular slave device has the advantage of requiring one less physical pin connection to implement pin sleep on SPI. It has the disadvantage of putting the device to sleep whenever the SPI master negates SPI_SSEL (meaning time is lost waiting for the device to wake), even if that was not the intent.

If the user has full control of SPI_SSEL so that it can control pin sleep, whether or not data needs to be transmitted, then sharing the pin may be a good option in order to make the SLEEP_REQUEST pin available for another purpose.

If the device is one of multiple slaves on the SPI, then the device sleeps while the SPI master talks to the other slave, but this is acceptable in most cases.

If you do not configure either pin as a peripheral, then the device stays awake, being unable to sleep in **SM**1 mode.

Configuration considerations

The configuration considerations are:

- How do you select the serial port? For example, should you use the UART or the SPI port?
- If you use the SPI port, what data format should you use in order to avoid processing invalid characters while transmitting?
- What SPI options do you need to configure?

Serial port selection

In the default configuration both the UART and SPI ports are configured for serial port operation. In this case, serial data goes out the UART until the host device asserts the SPI_SSEL signal. Thereafter all serial communications operate only on the SPI interface until a reset occurs.

If you enable only the UART, the XBee 868LP RF Module uses only the UART, and ignores the SPI_SSEL.

If you enable only the SPI, the XBee 868LP RF Module uses only the SPI, and ignores UART communications.

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

Data format

SPI only operates in API mode 1. The XBee 868LP RF Module does not support Transparent mode or API mode 2 (which escapes control characters). This means that the AP configuration only applies to the UART, and the device ignores it while using SPI.

SPI parameters

Most host processors with SPI hardware allow you to set the bit order, clock phase and polarity. For communication with all XBee 868LP RF Modules, the host processor must set these options as follows:

- Bit order: send MSB first
- Clock phase (CPHA): sample data on first (leading) edge
- Clock polarity (CPOL): first (leading) edge rises

All XBee 868LP RF Modules use SPI mode 0 and MSB first. Mode 0 means that data is sampled on the leading edge and that the leading edge rises. MSB first means that bit 7 is the first bit of a byte sent over the interface.

Serial buffers

To enable the UART port, DIN and DOUT must be configured as peripherals. To enable the SPI port, SPI_MISO, SPI_MOSI, SPI_SSEL, and SPI_CLK must be enabled as peripherals. If both ports are enabled, output goes to the UART until the first input on SPI. This is the default configuration.

When input occurs on either port, that port is selected as the active port and no input or output is allowed on the other port until the next reset of the module.

If you change the configuration to configure only one port, that port is the only one enabled or used. If the parameters are written with only one port enabled, the port that is not enabled is not used even temporarily after the next reset.

If both ports are disabled on reset, the device uses the UART regardless of the incorrect configuration to ensure that at least one serial port is operational.

Serial receive buffer

When serial data enters the device through the DIN pin (or the MOSI pin), it stores the data in the serial receive buffer until the device can process it. Under certain conditions, the device may not be able to process data in the serial receive buffer immediately. If large amounts of serial data are sent to the device such that the serial receive buffer would overflow, then it discards new data. If the UART is in use, you can avoid this by the host side honoring $\overline{\text{CTS}}$ flow control.

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

Serial transmit buffer

When the device receives RF data, it moves the data into the serial transmit buffer and sends it out the UART or SPI port. If the serial transmit buffer becomes full and the system buffers are also full, then it drops the entire RF data packet. Whenever the device receives data faster than it can process and transmit the data out the serial port, there is a potential of dropping data.

UART flow control

You can use the RTS and CTS pins to provide RTS and/or CTS flow control. CTS flow control provides an indication to the host to stop sending serial data to the device. RTS flow control allows the host to signal the device to not send data in the serial transmit buffer out the UART. To enable RTS/CTS flow control, use the **D6** and **D7** commands.

Note Serial port flow control is not possible when using the SPI port.

CTS flow control

If you enable CTS flow control (**D7** command), when the serial receive buffer is 17 bytes away from being full, the device de-asserts CTS (sets it high) to signal to the host device to stop sending serial data. The device reasserts CTS after the serial receive buffer has 34 bytes of space. See FT (Flow Control Threshold) for the buffer size.

In either case, CTS is not re-asserted until the serial receive buffer has **FT**-17 or less bytes in use.

RTS flow control

If you send the **D6** command to enable RTS flow control, the device does not send data in the serial transmit buffer out the DOUT pin as long as RTS is de-asserted (set high). Do not de-assert RTS for long periods of time or the serial transmit buffer will fill. If the device receives an RF data packet and the serial transmit buffer does not have enough space for all of the data bytes, it discards the entire RF data packet.

The UART Data Present Indicator is a useful feature when using RTS flow control. When enabled, the DIO19 line asserts (low asserted) when UART data is queued to be transmitted from the device. For more information, see P9 (SPI_ATTN).

If the device sends data out the UART when RTS is de-asserted (set high) the device could send up to five characters out the UART port after RTS is de-asserted.

Force UART operation

Condition

You configure a device with only the SPI enabled and no SPI master is available to access the SPI slave port

Solution

Use the following steps to recover the device to UART operation:

1. Hold the DIN/CONFIG low at reset time.
2. DIN/CONFIG forces a default configuration on the UART at 9600 baud and brings up the device in Command Mode on the UART port.

   You can send the appropriate commands to the device to configure it for UART operation.
3. If you write these parameters to the device, the module comes up with the UART enabled on the next reset.

Serial interface protocols

The XBee 868LP RF Module supports both Transparent and Application Programming Interface (API) serial interfaces.

Transparent operating mode

When operating in Transparent mode, the devices act as a serial line replacement. The device queues up all UART data received through the DIN pin for RF transmission. When RF data is received, the device sends the data out through the serial port. Use the Command mode interface to configure the device configuration parameters.

Note Transparent operation is not available when using SPI.

The device buffers data in the serial receive buffer and packetizes and transmits the data when it receives the following:

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

API operating mode

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

The API provides alternative means of configuring devices and routing data at the host application layer. A host application can send data frames to the device that contain address and payload information instead of using Command mode to modify addresses. The device sends 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:

- 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

Comparing Transparent and API modes

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

Feature	Description
Transparent mode features	
Simple interface	All received serial data is transmitted unless the device is in Command mode
Easy to support	It is easier for an application to support Transparent operation and Command mode
API mode features	
Easy to manage data transmissions to multiple destinations	Transmitting RF data to multiple remote devices only requires the application to change the address in the API frame. This process is much faster than in Transparent mode where the application must enter 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	Because acknowledgments are sent out of the serial interface, this provides more information about the health of the RF network and can be used to debug issues after the network has been deployed.
Received data frames indicate the sender's address	All received RF data API frames indicate the source address
Advanced addressing support	API transmit and receive frames can expose addressing fields including source and destination endpoints, cluster ID, and profile ID
Advanced networking diagnostics	API frames can provide indication of I/O 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
Simultaneous Commands	Query or set a configuration parameter while a pending command like **ND** is in progress. This cannot be done in Command mode. It is available in firmware versions 9009 or newer.

We recommend API mode 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 endpoints, cluster IDs, and/or profile IDs
- Uses the Device Profile services

API mode is required when:

- Receiving I/O samples from remote devices
- Using SPI for the serial port

If the conditions listed above do not apply (for example, 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

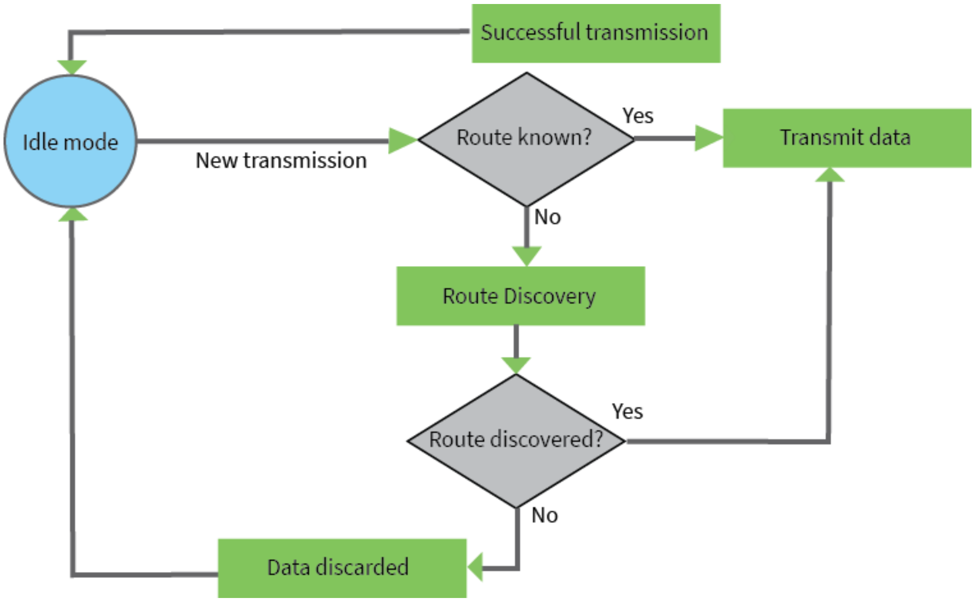
The XBee 868LP RF Module is in Receive Mode when it is not transmitting data. The device shifts into the other modes of operation under the following conditions:

- Transmit mode (Serial data in the serial receive buffer is ready to be packetized)
- Sleep mode
- Command Mode (Command mode sequence is issued (not available when using the SPI port))

Transmit mode

When the device receives serial data and is ready to packetize it, the device attempts to transmit the serial data. The destination address determines which node(s) will receive and send the data.

In the following diagram, route discovery applies only to DigiMesh transmissions. Once route discovery establishes a route, the device transmits the data. If route discovery fails to establish a route, the device discards the packet.



When DigiMesh data is transmitted from one node to another, the destination node transmits a network-level acknowledgment back across the established route to the source node. This acknowledgment packet indicates to the source node that the destination node received the data packet. If the source node does not receive a network acknowledgment, it retransmits the data.

For more information, see Data transmission and routing.

Receive mode

This is the default mode for the XBee 868LP RF Module. The device is in Receive mode when it is not transmitting data. If a destination node receives a valid RF packet, the destination node transfers the data to its serial transmit buffer.

Command mode

Command mode is a state in which the firmware interprets incoming characters as commands. It allows you to modify the device’s configuration using parameters you can set using AT commands. When you want to read or set any parameter of the XBee 868LP RF Module using this mode, you have to send an AT command. Every AT command starts with the letters **AT** followed by the two characters that identify the command and then by some optional configuration values.

The operating modes of the XBee 868LP RF Module are controlled by the AP (API Mode) setting, but Command mode is always available as a mode the device can enter while configured for any of the operating modes.

Command mode is available on the UART interface for all operating modes. You cannot use the SPI interface to enter Command mode.

Enter Command mode

To get a device to switch into Command mode, you must issue the following sequence: **+++** within one second. There must be at least one second preceding and following the **+++** sequence. Both the command character (**CC**) and the silence before and after the sequence (**GT**) are configurable. When the entrance criteria are met the device responds with **OK\r** on UART signifying that it has entered Command mode successfully and is ready to start processing AT commands.

If configured to operate in Transparent operating mode, when entering Command mode the XBee 868LP RF Module knows to stop sending data and start accepting commands locally.

Note Do not press **Return** or **Enter** after typing **+++** because it interrupts the guard time silence and prevents you from entering Command mode.

When the device is in Command mode, it listens for user input and is able to receive AT commands on the UART. If **CT** time (default is 10 seconds) passes without any user input, the device drops out of Command mode and returns to the previous operating mode. You can force the device to leave Command mode by sending CN (Exit Command Mode).

You can customize the command character, the guard times and the timeout in the device's configuration settings. For more information, see CC (Command Sequence Character), CT (Command Mode Timeout) and GT (Guard Times).

Troubleshooting

Failure to enter Command mode is often due to baud rate mismatch. Ensure that the baud rate of the connection matches the baud rate of the device. By default, BD (Baud Rate) = **3** (9600 b/s).

There are two alternative ways to enter Command mode:

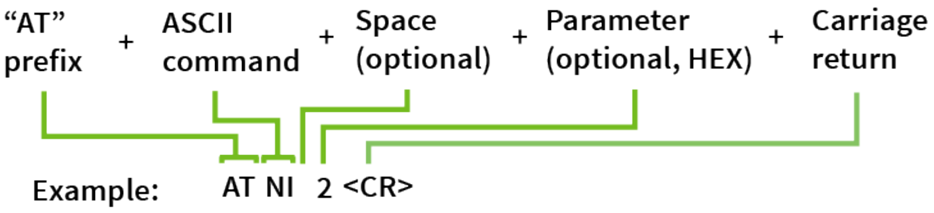
- A serial break for six seconds enters Command mode. You can issue the "break" command from a serial console, it is often a button or menu item.
- Asserting DIN (serial break) upon power up or reset enters Command mode. XCTU guides you through a reset and automatically issues the break when needed.

Both of these methods temporarily set the device's baud rate to 9600 and return an **OK** on the UART to indicate that Command mode is active. When Command mode exits, the device returns to normal operation at the baud rate that **BD** is set to.

Send AT commands

Once the device enters Command mode, use the syntax in the following figure to send AT commands. Every AT command starts with the letters **AT**, which stands for "attention." The AT is followed by two characters that indicate which command is being issued, then by some optional configuration values.

To read a parameter value stored in the device's register, omit the parameter field.



The preceding example changes NI (Node Identifier) to **My XBee**.

Multiple AT commands

You can send multiple AT commands at a time when they are separated by a comma in Command mode; for example, **ATNIMy XBee,AC<cr>**.

The preceding example changes the **NI (Node Identifier)** to **My XBee** and makes the setting active through AC (Apply Changes).

Parameter format

Refer to the list of AT commands for the format of individual AT command parameters. Valid formats for hexidecimal values include with or without a leading **0x** for example **FFFF** or **0xFFFF**.

Response to AT commands

When using AT commands to set parameters the XBee 868LP RF Module responds with **OK<cr>** if successful and **ERROR<cr>** if not.

Apply command changes

Any changes you make to the configuration command registers using AT commands do not take effect until you apply the changes. For example, if you send the **BD** command to change the baud rate, the actual baud rate does not change until you apply the changes. To apply changes:

1. Send AC (Apply Changes).
2. Send WR (Write.
   or:
3. Exit Command mode.

Make command changes permanent

Send a WR (Write command to save the changes. **WR** writes parameter values to non-volatile memory so that parameter modifications persist through subsequent resets.

Send as RE (Restore Defaults) to wipe settings saved using **WR** back to their factory defaults.

Note You still have to use **WR** to save the changes enacted with **RE**.

Exit Command mode

1. Send CN (Exit Command Mode) followed by a carriage return.
   or:
2. If the device does not receive any valid AT commands within the time specified by CT (Command Mode Timeout), it returns to Transparent or API mode. The default Command mode timeout is 10 seconds.

For an example of programming the device using AT Commands and descriptions of each configurable parameter, see AT commands.

Sleep mode

Sleep modes allow the device to enter states of low power consumption when not in use. The XBee 868LP RF Module supports both pin sleep (Sleep mode entered on pin transition) and cyclic sleep (device sleeps for a fixed time).

Sleep modes allow the device to enter states of low power consumption when not in use. XBee devices support both pin sleep, where the device enters sleep mode upon pin transition, and cyclic sleep, where the device sleeps for a fixed time. For more information, see Sleep modes.

Sleep modes

About sleep modes

A number of low-power modes exist to enable devices to operate for extended periods of time on battery power. Use the **SM** command to enable these sleep modes. The sleep modes are characterized as either:

- Asynchronous (**SM** = 1, 4, 5).
- Synchronous (**SM** = 7, 8).

Asynchronous modes

- Do not use asynchronous sleep modes in a synchronous sleeping network, and vice versa.
- Use the asynchronous sleep modes to control the sleep state on a device by device basis.
- Do not use devices operating in asynchronous sleep mode to route data.
- We strongly encourage you to set asynchronous sleeping devices as end-devices using the **CE** command. This prevents the node from attempting to route data.

Synchronous modes

Synchronous sleep makes it possible for all nodes in the network to synchronize their sleep and wake times. All synchronized cyclic sleep nodes enter and exit a low power state at the same time.

This forms a cyclic sleeping network.

- A device acting as a sleep coordinator sends a special RF packet called a sync message to synchronize nodes.
- To make a device in the network a coordinator, a node uses several resolution criteria through a process called nomination.
- The sleep coordinator sends one sync message at the beginning of each wake period. The coordinator sends the sync message as a broadcast and every node in the network repeats it.
- You can change the sleep and wake times for the entire network by locally changing the settings on an individual device. The network uses the most recently set sleep settings.

Normal mode

Set **SM** to 0 to enter Normal mode.

Normal mode is the default sleep mode. If a device is in this mode, it does not sleep and is always awake.

Use mains-power for devices in Normal mode.

A device in Normal mode synchronizes to a sleeping network, but does not observe synchronization data routing rules; it routes data at any time, regardless of the network's wake state.

When synchronized, a device in Normal mode relays sync messages that sleep-compatible nodes generate, but does not generate sync messages itself.

Once a device in Normal mode synchronizes with a sleeping network, you can put it into a sleep-compatible sleep mode at any time.

Asynchronous pin sleep mode

Set **SM** to 1 to enter asynchronous pin sleep mode.

Pin sleep allows the device to sleep and wake according to the state of the SLEEP_RQ pin (pin 10).

When you assert SLEEP_RQ (high), the device finishes any transmit or receive operations and enters a low-power state.

When you de-assert SLEEP_RQ (low), the device wakes from pin sleep.

Asynchronous cyclic sleep mode

Set **SM** to 4 to enter asynchronous cyclic sleep mode.

Cyclic sleep allows the device to sleep for a specific time and wake for a short time to poll.

If the device receives serial or RF data while awake, it extends the time before it returns to sleep by the specific amount the **ST** command provides. Otherwise, it enters sleep mode immediately.

The ON_SLEEP line asserts (high) when the device wakes, and is de-asserted (low) when the device sleeps.

If you use the **D7** command to enable hardware flow control, the CTS pin asserts (low) when the device wakes and can receive serial data, and de-asserts (high) when the device sleeps.

Asynchronous cyclic sleep with pin wake up mode

Set **SM** to **5** to enter Asynchronous cyclic sleep with pin wake up mode.

This mode is a slight variation on (**SM** = **4**) that allows the device to wake prematurely by asserting the SLEEP_RQ pin (pin 9). In (**SM** = 5), the device wakes after the sleep period expires, or if a high-to-low transition occurs on the SLEEP_RQ pin.

Synchronous sleep support mode

Set **SM** to 7 to enter synchronous sleep support mode.

A device in synchronous sleep support mode synchronizes itself with a sleeping network but will not itself sleep. At any time, the device responds to new devices that are attempting to join the sleeping network with a sync message. A sleep support device only transmits normal data when the other devices in the sleeping network are awake. You can use sleep support devices as preferred sleep coordinator devices and as aids in adding new devices to a sleeping network.

Synchronous cyclic sleep mode

Set **SM** to 8 to enter synchronous cyclic sleep mode.

A device in synchronous cyclic sleep mode sleeps for a programmed time, wakes in unison with other nodes, exchanges data and sync messages, and then returns to sleep. While asleep, it cannot receive RF messages or receive data (including commands) from the UART port.

Generally, the network’s sleep coordinator specifies the sleep and wake times based on its **SP** and **ST** settings. The device only uses these parameters at startup until the device synchronizes with the network.

When a device has synchronized with the network, you can query its sleep and wake times with the **OS** and **OW** commands respectively.

If **D9** = 1 (ON_SLEEP enabled) on a cyclic sleep node, the ON_SLEEP line asserts when the device is awake and de-asserts when the device is asleep.

If **D7** = 1, the device de-asserts CTS while asleep.

A newly-powered, unsynchronized, sleeping device polls for a synchronized message and then sleeps for the period that the **SP** command specifies, repeating this cycle until it synchronizes by receiving a sync message. Once it receives a sync message, the device synchronizes itself with the network.

Note Configure all nodes in a synchronous sleep network to operate in either synchronous sleep support mode or synchronous cyclic sleep mode. asynchronous sleeping nodes are not compatible with synchronous sleeping nodes.

Wake timer

In asynchronous cyclic sleep mode (**SM** = **4** or **SM** = **5**), if a device receives serial or RF data, it starts a sleep timer (time until sleep). Any data received serially or by RF link resets the timer. Use ST (Wake Time) to set the timer duration. While the device is awake, it sends regular poll requests to its parent to check for buffered data. If the RF data rate is 80 kb/s (**BR** = **1**), the poll occurs every 100 ms. Otherwise, (**BR** = **0**), the poll occurs every 300 ms. The device returns to sleep when the sleep timer expires.

Indirect messaging and polling

To enable reliable communication with sleeping devices, you can use the **CE** (Routing/Messaging Mode) command to enable indirect messaging and polling.

Indirect messaging

Indirect messaging is a communication mode designed for communicating with asynchronous sleeping devices. A device can enable indirect messaging by making itself an indirect messaging coordinator with the **CE** command. An indirect messaging coordinator does not immediately transmit a P2MP unicast when it is received over the serial port. Instead the device holds onto the data until it is requested via a poll. On receiving a poll, the indirect messaging coordinator sends a queued data packet (if available) to the requestor.

Because it is possible for a polling device to be eliminated, a mechanism is in place to purge unrequested data packets. If the coordinator holds an indirect data packet for an indirect messaging poller for more than 2.5 times its **SP** value, then the packet is purged. We suggest setting the **SP** of the coordinator to the same value as the highest **SP** time that exists among the pollers in the network. If the coordinator is in API mode, a TxStatus message is generated for a purged data packet with a status of 0x75 (**INDIRECT_MESSAGE_UNREQUESTED**).

An indirect messaging coordinator queues up as many data packets as it has buffers available. After the coordinator uses all of its available buffers, it holds transmission requests unprocessed on the serial input queue. After the serial input queue is full, the device de-asserts CTS (if hardware flow control is enabled). After receiving a poll or purging data from the indirect messaging queue the buffers become available again.

Indirect messaging only functions with P2MP unicast messages. Indirect messaging has no effect on P2MP broadcasts, directed broadcasts, repeater packets, or DigiMesh packets. These messages are sent immediately when received over the serial port and are not put on the indirect messaging queue.

Polling

Polling is the automatic process by which a node can request data from an indirect messaging coordinator. To enable polling on a device, configure it as an end device with the **CE** command. When you enable polling, the device sends a poll request a minimum of once every 100 ms. When the device sends normal data to the destination specified by the **DH**/**DL** of end device module, the data also functions as a poll.

When a polling device is also an asynchronous sleeping device, that device sends a poll shortly after waking from sleep. After that first poll is sent, the device sends polls in the normal manner described previously until it returns to sleep.

Sleeping routers

The Sleeping Router feature of DigiMesh makes it possible for all nodes in the network to synchronize their sleep and wake times. All synchronized cyclic sleep nodes enter and exit a low power state at the same time. This forms a cyclic sleeping network.

Devices synchronize by receiving a special RF packet called a sync message sent by a device acting as a sleep coordinator. A device in the network becomes a sleep coordinator through a process called nomination. The sleep coordinator sends one sync message at the beginning of each wake period. The device sends a sync message as a broadcast that is repeated by every device in the network. To change the sleep and wake times for the entire network, change the settings on an individual node locally. The network uses the most recently set sleep settings.

For more information, see Become a sleep coordinator.

Sleep coordinator sleep modes in the DigiMesh network

In a synchronized sleeping network, one node acts as the sleep coordinator. During normal operations, at the beginning of a wake cycle the sleep coordinator sends a sync message as a broadcast to all nodes in the network. This message contains synchronization information and the wake and sleep times for the current cycle. All cyclic sleep nodes that receive a sync message remain awake for the wake time and then sleep for the specified sleep period.

The sleep coordinator sends one sync message at the beginning of each cycle with the current wake and sleep times. All router nodes that receive this sync message relay the message to the rest of the network. If the sleep coordinator does not hear a rebroadcast of the sync message by one of its immediate neighbors, then it re-sends the message one additional time.

If you change the **SP** or **ST** parameters, the network does not apply the new settings until the beginning of the next wake time. For more information, see Change sleep parameters.

A sleeping router network is robust enough that an individual node can go several cycles without receiving a sync message, due to RF interference, for example. As a node misses sync messages, the time available for transmitting messages during the wake time reduces to maintain synchronization accuracy. By default, a device reduces its active sleep time progressively as it misses sync messages.

Synchronization messages

A sleep coordinator regularly sends sync messages to keep the network in sync. Unsynchronized nodes also send messages requesting sync information.

Sleep compatible nodes use Deployment mode when they first power up and the sync message has not been relayed. A sleep coordinator in Deployment mode rapidly sends sync messages until it receives a relay of one of those messages. Deployment mode:

- Allows you to effectively deploy a network.
- Allows a sleep coordinator that resets to rapidly re-synchronize with the rest of the network.

If a node exits deployment mode and then receives a sync message from a sleep coordinator that is in Deployment mode, it rejects the sync message and sends a corrective sync to the sleep coordinator.

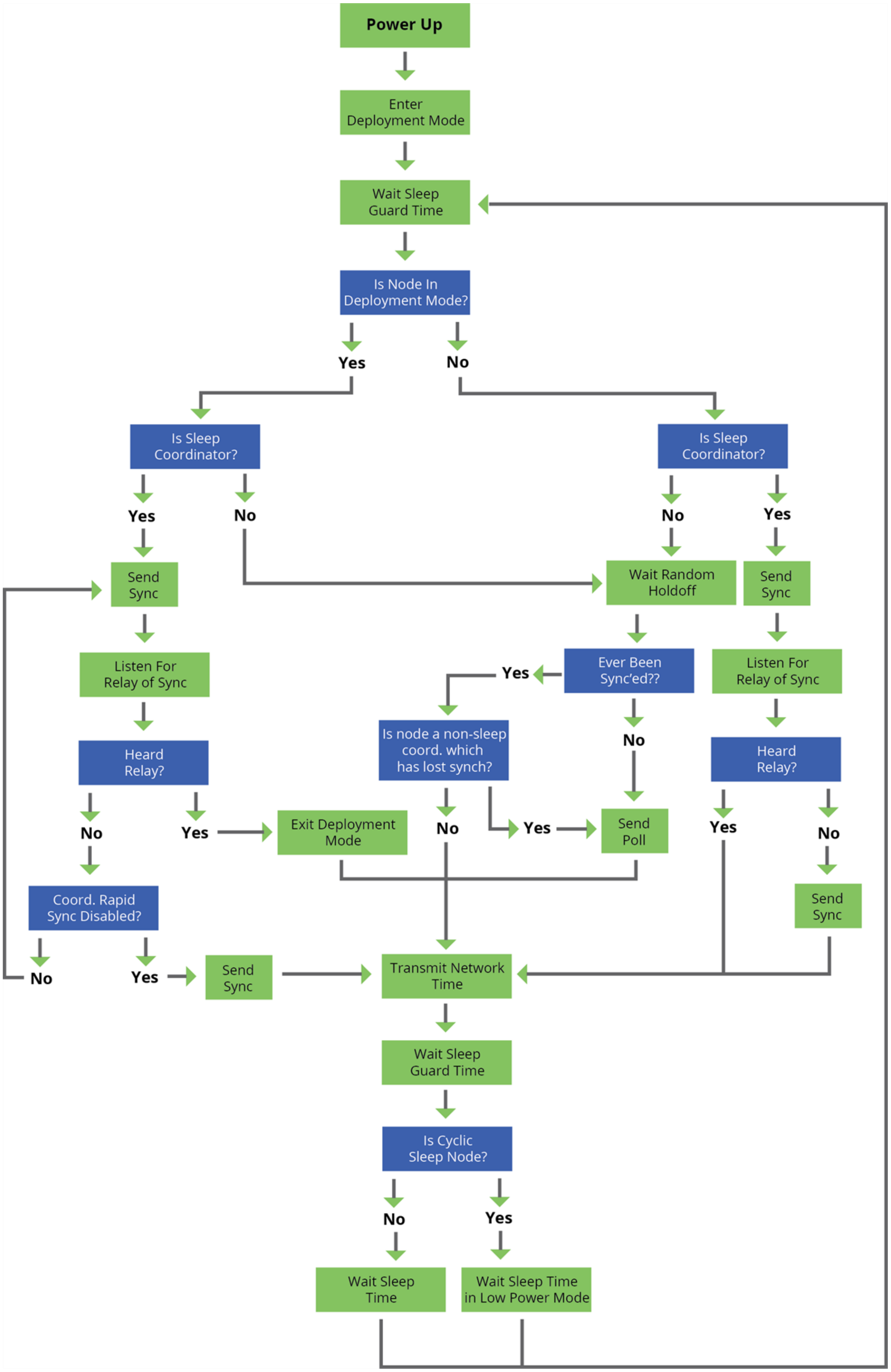
Use the **SO** (sleep options) command to disable deployment mode. This option is enabled by default.

A sleep coordinator that is not in deployment mode sends a sync message at the beginning of the wake cycle. The sleep coordinator listens for a neighboring node to relay the sync. If it does not hear the relay, the sleep coordinator sends the sync one additional time.

A node that is not a sleep coordinator and has never been synchronized sends a message requesting sync information at the beginning of its wake cycle. Synchronized nodes which receive one of these messages respond with a synchronization packet.

If you use the **SO** command to configure nodes as non-coordinators, and if the non-coordinators go six or more sleep cycles without hearing a sync, they send a message requesting sync at the beginning of their wake period.

The following diagram illustrates the synchronization behavior of sleep compatible devices.


Power Up
Enter Deployment Mode
Wait Sleep Guard Time
Is Node In Deployment Mode?
Yes
No
Is Sleep Coordinator?
Yes
No
Is Sleep Coordinator?
No
Yes
Send Sync
Wait Random Holdoff
Send Sync
Listen For Relay of Sync
Yes
Ever Been Sync'ed??
Listen For Relay of Sync
Is node a non-sleep coord. which has lost synch?
No
Heard Relay?
Heard Relay?
No
Yes
Exit Deployment Mode
No
Yes
Send Poll
Yes
No
Coord. Rapid Sync Disabled?
Send Sync
No
Yes
Send Sync
Transmit Network Time
Wait Sleep Guard Time
Is Cyclic Sleep Node?
No
Yes
Wait Sleep Time
Wait Sleep Time in Low Power Mode

Become a sleep coordinator

In DigiMesh networks, a device can become a sleep coordinator in one of four ways:

- Define a preferred sleep coordinator
- A potential sleep coordinator misses three or more sync messages
- Press the Commissioning Pushbutton twice on a potential sleep coordinator
- Change the sleep timing values on a potential sleep coordinator

Preferred sleep coordinator option

You can specify that a node always act as a sleep coordinator. To do this, set the preferred sleep coordinator bit (bit 0) in the **SO** command to 1.

A node with the sleep coordinator bit set always sends a sync message at the beginning of a wake cycle. To avoid network congestion and synchronization conflicts, do not set this bit on more than one node in the network.

A node that is centrally located in the network can serve as a good sleep coordinator, because it minimizes the number of hops a sync message takes to get across the network.

A sleep support node and/or a node that is mains powered is a good candidate to be a sleep coordinator.

CAUTION! Use the preferred sleep coordinator bit with caution. The advantages of using the option become weaknesses if you use it on a node that is not in the proper position or configuration. Also, it is not valid to have the sleep coordinator option bit set on more than one node at a time.

You can also use the preferred sleep coordinator option when you set up a network for the first time. When you start a network, you can configure a node as a sleep coordinator so it will begin sending sleep messages. After you set up the network, it is best to disable the preferred sleep coordinator bit.

Resolution criteria and selection option

There is an optional selection process with resolution criteria that occurs on a node if it loses contact with the network sleep coordinator. By default, this process is disabled. Use the **SO** command to enable this process. This process occurs automatically if a node loses contact with the previous sleep coordinator.

If you enable the process on any sleep compatible node, it is eligible to become the sleep coordinator for the network.

A sleep compatible node may become a sleep coordinator if it:

- Misses three or more sync messages.
- It is not configured as a non-coordinator by setting bit 1 of **SO**.

If such a node wins out in the selection process, it becomes the new network sleep coordinator.

It is possible for multiple nodes to declare themselves as the sleep coordinator. If this occurs, the firmware uses the following resolution criteria to identify the sleep coordinator from among the nodes using the selection process:

1. Newer sleep parameters always take priority over older sleep parameters. The age of the sleep parameters is determined by a sequence number that increments when an overriding sync is

sent.

2. Otherwise, the node with the preferred sleep coordinator bit set takes precedence.
3. Otherwise, a sleep support node—**SM 7**—takes priority over a node that is not a sleep support node—**SM 8**.
4. Otherwise, the node with highest serial number becomes the sleep coordinator.

Commissioning Pushbutton option

Use the Commissioning Pushbutton to select a device to act as the sleep coordinator.

If you enable the Commissioning Pushbutton functionality, you can immediately select a device as a sleep coordinator by pressing the Commissioning Pushbutton twice or by issuing the **CB2** command. The device you select in this manner is still subject to the resolution criteria process.

Only potential sleep coordinator nodes honor Commissioning Pushbutton nomination requests. A node configured as a non-sleep coordinator ignores commissioning button nomination requests.

Overriding syncs

Any sleep compatible node in the network that does not have the non-coordinator sleep option set can send an overriding sync and become the network sleep coordinator. An overriding sync effectively changes the synchronization of all nodes in the network to the **ST** and **SP** values of the node sending the overriding sync. It also selects the node sending the overriding sync as the network sleep coordinator. While this is a powerful operation, it may be an undesired side effect because the current sleep coordinator may have been carefully selected and it is not desired to change it. Additionally the current wake and sleep cycles may be desired rather than the parameters on the node sending the overriding sync. For this reason, it is important to know what kicks off an overriding sync.

An overriding sync occurs whenever **ST** or **SP** is changed to a value different than **OW** or **OS** respectively. For example no overriding sync will occur if **SP** is changed from **190** to **C8** if the network was already operating with **OS** at **C8**. On the other hand, if **SP** is changed from **190** to **190**—meaning no change—and **OS** is **C8**, than an overriding sync will occur because the network parameters are being changed.

Even parameters that seem unrelated to sleep can kick off an overriding sync. These are **NH**, **NN**, **RN**, and **MT**. When any of these parameters are changed, they can affect network traversal time. If such changes cause the configured value of **ST** to be smaller than the value needed for network traversal, then **ST** is increased and if that increased value is different than **OW**, then an overriding sync will occur.

For most applications, we recommend configuring the **NH**, **NN**, **RN**, and **MT** network parameters during initial deployment only. The default values of **NH** and **NN** are optimized to work for most deployments. Additionally, it would be best to set **ST** and **SP** the same on all nodes in the network while keeping **ST** sufficiently large so that it will not be affected by an inadvertent change of **NH**, **NN**, **RN**, or **MT**.

Sleep guard times

To compensate for variations in the timekeeping hardware of the various devices in a sleeping router network, the network allocates sleep guard times at the beginning and end of the wake period. The size of the sleep guard time varies based on the sleep and wake times you select and the number of sleep cycles that elapse since receiving the last sync message. The sleep guard time guarantees that a destination module will be awake when the source device sends a transmission. As a node misses more and more consecutive sync messages, the sleep guard time increases in duration and decreases the available transmission time.

Auto-early wake-up sleep option

Similar to the sleep guard time, the auto early wake-up option decreases the sleep period based on the number of sync messages a node misses. This option comes at the expense of battery life.

Use the **SO** command to disable auto-early wake-up sleep. This option is enabled by default.

Select sleep parameters

Choosing proper sleep parameters is vital to creating a robust sleep-enabled network with a desirable battery life. To select sleep parameters that will be good for most applications, follow these steps:

1. Choose **NH**.

   Based on the placement of the nodes in your network, select the appropriate values for the **NH** (Network Hops) parameter .

   We optimize the default values of **NH** to work for the majority of deployments. In most cases, we suggest that you do not modify these parameters from their default values. Decreasing these parameters for small networks can improve battery life, but take care to not make the values too small.
2. Calculate the Sync Message Propagation Time (SMPT).

   This is the maximum amount of time it takes for a sleep synchronization message to propagate to every node in the network. You can estimate this number with the following formula:
   SMPT = **NH***(**MT**+1)*18 ms.
3. Select the duty cycle you want.
   The ratio of sleep time to wake time is the factor that has the greatest effect on the device's power consumption. Battery life can be estimated based on the following factors:
   - sleep period
   - wake time
   - sleep current
   - RX current
   - TX current
   - battery capacity
4. Choose the sleep period and wake time.

   The wake time must be long enough to transmit the desired data as well as the sync message. The **ST** parameter automatically adjusts upwards to its minimum value when you change other AT commands that affect it (**SP**, and **NH**).

   Use a value larger than this minimum. If a device misses successive sync messages, it reduces its available transmit time to compensate for possible clock drift. Budget a large enough **ST** time to allow for the device to miss a few sync messages and still have time for normal data transmissions.

Start a sleeping synchronous network

By default, all new nodes operate in normal (non-sleep) mode. To start a synchronous sleeping network, follow these steps:

1. Set **SO** to 1 to enable the preferred sleep coordinator option on one of the nodes.
2. Set its **SM** to a synchronous sleep compatible mode (7 or 8) with its **SP** and **ST** set to a quick cycle time. The purpose of a quick cycle time is to allow the network to send commands quickly through the network during commissioning.
3. Power on the new nodes within range of the sleep coordinator. The nodes quickly receive a sync message and synchronize themselves to the short cycle **SP** and **ST** set on the sleep coordinator.
4. Configure the new nodes to the sleep mode you want, either cyclic sleeping modes or sleep support modes.
5. Set the **SP** and **ST** values on the sleep coordinator to the values you want for the network.
6. In order to reduce the possibility of an unintended overriding sync, set **SP** and **ST** to the intended sleep/wake cycle on all nodes in the network. Be sure that **ST** is large enough to prevent it from being inadvertently increased by changing **NN**, **NH**, or **MT**.
7. Wait a sleep cycle for the sleeping nodes to sync themselves to the new **SP** and **ST** values.
8. Disable the preferred sleep coordinator option bit on the sleep coordinator unless you want a preferred sleep coordinator.
9. Deploy the nodes to their positions.

Alternatively, prior to deploying the network you can use the **WR** command to set up nodes with their sleep settings pre-configured and written to flash. If this is the case, you can use the Commissioning Pushbutton and associate LED to aid in deployment:

1. If you are going to use a preferred sleep coordinator in the network, deploy it first.
2. If there will not be a preferred sleep coordinator, select a node for deployment, power it on and press the Commissioning Pushbutton twice. This causes the node to begin emitting sync messages.
3. Verify that the first node is emitting sync messages by watching its associate LED. A slow blink indicates that the node is acting as a sleep coordinator.
4. Power on nodes in range of the sleep coordinator or other nodes that have synchronized with the network. If the synchronized node is asleep, you can wake it by pressing the Commissioning Pushbutton once.
5. Wait a sleep cycle for the new node to sync itself.
6. Verify that the node syncs with the network. The associate LED blinks when the device is awake and synchronized.
7. Continue this process until you deploy all of the nodes.

Add a new node to an existing network

To add a new node to the network, the node must receive a sync message from a node already in the network. On power-up, an unsynchronized, sleep compatible node periodically sends a broadcast requesting a sync message and then sleeps for its **SP** period. Any node in the network that receives this message responds with a sync. Because the network can be asleep for extended periods of time, and cannot respond to requests for sync messages, there are methods you can use to sync a new node while the network is asleep.

1. Power the new node on within range of a sleep support node. Sleep support nodes are always awake and able to respond to sync requests promptly.

2. You can wake a sleeping cyclic sleep node in the network using the Commissioning Pushbutton. Place the new node in range of the existing cyclic sleep node. Wake the existing node by holding down the Commissioning Pushbutton for two seconds, or until the node wakes. The existing node stays awake for 30 seconds and responds to sync requests while it is awake.

If you do not use one of these two methods, you must wait for the network to wake up before adding the new node.

Place the new node in range of the network with a sleep/wake cycle that is shorter than the wake period of the network.

The new node periodically sends sync requests until the network wakes up and it receives a sync message.

Change sleep parameters

To change the sleep and wake cycle of the network, select any sleep coordinator capable node in the network and change the **SP** and/or **ST** of the node to values different than those the network currently uses.

- If you use a preferred sleep coordinator or if you know which node acts as the sleep coordinator, we suggest that you use this node to make changes to network settings.
- If you do not know the network sleep coordinator, you can use any node that does not have the non-sleep coordinator sleep option bit set. For details on the bit, see SO (Sleep Options).

When you make changes to a node’s sleep parameters, that node becomes the network’s sleep coordinator unless it has the non-sleep coordinator option selected. It sends a sync message with the new sleep settings to the entire network at the beginning of the next wake cycle. The network immediately begins using the new sleep parameters after it sends this sync.

Changing sleep parameters increases the chances that nodes will lose sync. If a node does not receive the sync message with the new sleep settings, it continues to operate on its old settings. To minimize the risk of a node losing sync and to facilitate the re-syncing of a node that does lose sync, take the following precautions:

1. Whenever possible, avoid changing sleep parameters.
2. Enable the missed sync early wake up sleep option in the **SO** command. This option is enabled by default. This command tells a node to wake up progressively earlier based on the number of cycles it goes without receiving a sync. This increases the probability that the un-synced node will be awake when the network wakes up and sends the sync message.

Note Using this sleep option increases reliability but may decrease battery life. Nodes using this sleep option that miss sync messages increase their wake time and decrease their sleep time during cycles where they miss the sync message. This increases power consumption.

When you are changing between two sets of sleep settings, choose settings so that the wake periods of the two sleep settings occur at the same time. In other words, try to satisfy the following equation:

$$(SP_1 + ST_1) = N * (SP_2 + ST_2)$$

where SP_1/ST_1 and SP_2/ST_2 are the desired sleep settings and N is an integer.

Rejoin nodes that lose sync

DigiMesh networks get their robustness from routing redundancies which may be available. We recommend architecting the network with redundant mesh nodes to increase robustness.

If a scenario exists where the only route connecting a subnet to the rest of the network depends on a single node, and that node fails or the wireless link fails due to changing environmental conditions (a catastrophic failure condition), then multiple subnets may arise using the same wake and sleep intervals. When this occurs the first task is to repair, replace, and strengthen the weak link with new and/or redundant devices to fix the problem and prevent it from occurring in the future.

When you use the default DigiMesh sleep parameters, separated subnets do not drift out of phase with each other. Subnets can drift out of phase with each other if you configure the network in one of the following ways:

- If you disable the non-sleep coordinator bit in the **SO** command on multiple devices in the network, they are eligible for the network to nominate them as a sleep coordinator. For more details, see SO (Sleep Options).
- If the devices in the network do not use the auto early wake-up sleep option.

If a network has multiple subnets that drift out of phase with each other, get the subnets back in phase with the following steps:

1. Place a sleep support node in range of both subnets.
2. Select a node in the subnet that you want the other subnet to sync with.
3. Use this node to slightly change the sleep cycle settings of the network, for example, increment **ST**.
4. Wait for the subnet's next wake cycle. During this cycle, the node you select to change the sleep cycle parameters sends the new settings to the entire subnet it is in range of, including the sleep support node that is in range of the other subnet.
5. Wait for the out of sync subnet to wake up and send a sync. When the sleep support node receives this sync, it rejects it and sends a sync to the subnet with the new sleep settings.
6. The subnets will now be in sync. You can remove the sleep support node.
7. You can also change the sleep cycle settings back to the previous settings.

If you only need to replace a few nodes, you can use this method:

1. Reset the out of sync node and set its sleep mode to Synchronous Cyclic Sleep mode (**SM** = 8).
2. Set up a short sleep cycle.
3. Place the node in range of a sleep support node or wake a sleeping node with the Commissioning Pushbutton.
4. The out of sync node receives a sync from the node that is synchronized to the network. It then syncs to the network sleep settings.

Diagnostics

The following diagnostics are useful in applications that manage a sleeping router network:

Query sleep cycle

Use the **OS** and **OW** commands to query the current operational sleep and wake times that a device uses.

Sleep status

Use the **SS** command to query useful information regarding the sleep status of the device. Use this command to query if the node is currently acting as a network sleep coordinator.

Missed sync messages command

Use the **MS** command to query the number of cycles that elapsed since the device received a sync message.

Sleep status API messages

When you use the **SO** command to enable this option, a device that is in API operating mode outputs modem status frames immediately after it wakes up and prior to going to sleep.

Advanced application features

Remote configuration commands

The API firmware has provisions to send configuration commands to remote devices using the Remote Command Request API frame (see API operation). Use the API frame to send commands to a remote device to read or set command parameters.

Send a remote command

To send a remote command, populate the Remote Command Request frame with:

- 64-bit address of the remote device
- Correct command options value
- Command and parameter data (optional)

If you want a command response, set the Frame ID set to a non-zero value. Only unicasts of remote commands are supported, and remote commands cannot be broadcast.

Apply changes on remote devices

When you use remote commands to change command parameter settings on a remote device, parameter changes do not take effect until you apply the changes. For example, changing the **BD** parameter does not change the serial interface on the remote until the changes are applied. To apply changes, do one of the following:

Remote command responses

If the remote device receives a remote command request transmission, and the API frame ID is non-zero, the remote sends a remote command response transmission back to the device that sent the remote command. When a remote command response transmission is received, a device sends a remote command response API frame out its serial port. The remote command response indicates the status of the command (success, or reason for failure), and in the case of a command query, it includes the register value. The device that sends a remote command will not receive a remote command response frame if either of the following conditions exist:

- The destination device could not be reached.
- The frame ID in the remote command request is set to 0.

Network commissioning and diagnostics

We call the process of discovering and configuring devices in a network for operation, "network commissioning." Devices include several device discovery and configuration features. In addition to configuring devices, you must develop a strategy to place devices to ensure reliable routes. To accommodate these requirements, modules include features to aid in placing devices, configuring devices, and network diagnostics.

Configure devices

You can configure XBee devices 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.

Network link establishment and maintenance

Build aggregate routes

In many applications it is necessary for many or all of the nodes in the network to transmit data to a central aggregator node. In a new DigiMesh network the overhead of these nodes discovering routes to the aggregator node can be extensive and taxing on the network. To eliminate this overhead, use the **AG** command to automatically build routes to an aggregate node in a DigiMesh network.

Send a unicast

To send a unicast, devices configured for Transparent mode (**AP** = **0**) must set their **DH**/**DL** registers to the MAC address of the node which they need to transmit to. In networks of Transparent mode devices which transmit to an aggregator node, it is necessary to set every device's **DH**/**DL** registers to the MAC address of the aggregator node. Use the **AG** command to set the **DH**/**DL** registers of all the nodes in a DigiMesh network to that of the aggregator node.

Use the AG command

Upon deploying a DigiMesh network, send the **AG** command on the desired aggregator node to cause all nodes in the network to build routes to the aggregator node. You can use the command to automatically update the **DH**/**DL** registers to match the MAC address of the aggregator node.

The **AG** command requires a 64-bit parameter. The parameter indicates the current value of the **DH**/**DL** registers on a device which should be replaced by the 64-bit address of the node sending the **AG** broadcast. If it is not desirable to update the **DH/DL** of the device receiving the **AG** broadcast, you can use the invalid address of 0xFFFE. API enabled devices output an Aggregate Addressing Update - 0x8E if they update their **DH**/**DL** address.

All devices that receive an **AG** broadcast update their routing table information to build a route to the sending device, regardless of whether or not their **DH**/**DL** address is updated. This routing information will be used for future transmissions of DigiMesh unicasts.

Example 1: To update the **DH**/**DL** registers of all modules in the network to be equal to the MAC address of an aggregator node with a MAC address of **0x0013a2004052c507** after network deployment the following technique could be employed:

1. Deploy all devices in the network with the default **DH**/**DL** of 0xFFFF.
2. Send an **ATAGFFFF** command on the aggregator node.

Following the preceding sequence would result in all of the nodes in the network which received the **AG** broadcast to have a **DH** of **0x0013a200** and a **DL** of **0x4052c507**. These nodes would have automatically built a route to the aggregator.

Example 2: To cause all nodes in the network to build routes to an aggregator node with a MAC address of **0x0013a2004052c507** without affecting the **DH**/**DL** of any nodes in the network, send the **AGFFFE** command on the aggregator node. This sends an **AG** broadcast to all nodes in the network.

All of the nodes will update their internal routing table information to contain a route to the aggregator node. None of the nodes update their **DH**/**DL** registers, because none of the registers are set to an address of **0xFFFE**.

Node replacement

You can also use the AG command to update the routing table and **DH**/**DL** registers in the network after a device is replaced, and you can update the **DH**/**DL** registers of nodes in the network.

- To update only the routing table information without affecting the **DH**/**DL** registers, use Example 2.
- To update the **DH**/**DL** registers of the network, use the method in the following example.

Example: Use the device with serial number 0x0013a2004052c507 as a network aggregator and replace it with a device with serial number 0x0013a200f5e4d3b2. Issue the AG0013a2004052c507 command on the new module. This causes all devices with a **DH**/**DL** register setting of 0x0013a2004052c507 to update their **DH**/**DL** register setting to the MAC address of the sending device (0x0013a200f5e4d3b2).

Place devices

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

RSSI indicators

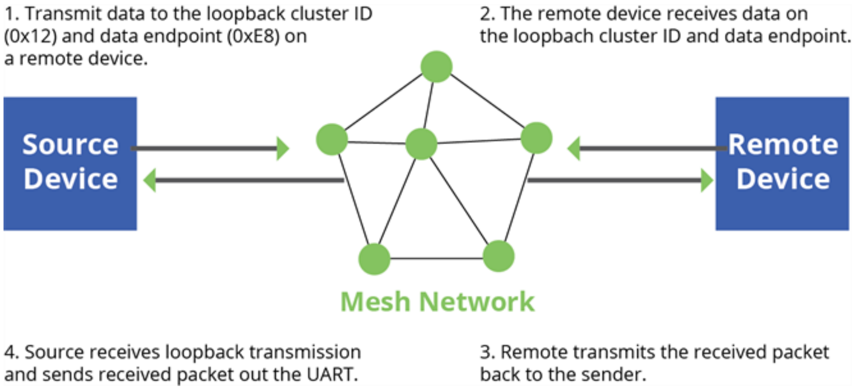
It is possible to measure the received signal strength on a device using the **DB** command. **DB** returns the RSSI value (measured in -dBm) of the last received packet. However, this number can be misleading in DigiMesh networks. The **DB** value only indicates the received signal strength of the last 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.

Determine the **DB** value in hardware using the RSSI/PWM device pin (pin 7). If you enable the RSSI PWM functionality (**P0** command), when the device receives data, it sets the RSSI PWM to a value based on the RSSI of the received packet (this value only indicates the quality of the last hop). You could connect this pin to an LED to indicate if the link is stable or not.

Test links in a network - loopback cluster

To measure the performance of a network, you can send unicast data through the network from one device to another to determine the success rate of several transmissions. To simplify link testing, the devices support a Loopback cluster ID (0x12) on the data endpoint (0xE8). The cluster ID on the data endpoint sends any data transmitted to it back to the sender.

The following figure demonstrates how you can use the Loopback cluster ID and data endpoint to measure the link quality in a mesh network.



1. Transmit data to the loopback cluster ID (0x12) and data endpoint (0xE8) on a remote device.

2. The remote device receives data on the loopbach cluster ID and data endpoint.

4. Source receives loopback transmission and sends received packet out the UART.

3. Remote transmits the received packet back to the sender.

The configuration steps for sending data to the loopback cluster ID depend on what mode the device is in. For details on setting the mode, see AP (API Mode). The following sections list the steps based on the device's mode.

Transparent operating mode configuration (AP = 0)

To send data to the loopback cluster ID on the data endpoint of a remote device:

1. Set the **CI** command to **0x12**.
2. Set the **SE** and **DE** commands to **0xE8** (default value).
3. Set the **DH** and **DL** commands to the address of the remote (**0** for the coordinator, or the 64-bit address of the remote).

After exiting Command mode, the device transmits any serial characters it received to the remote device, which returns those characters to the sending device.

API operating mode configuration (AP = 1 or AP = 2)

Send an Explicit Addressing Command Request - 0x11 using **0x12** as the cluster ID and **0xE8** as both the source and destination endpoint.

The remote device echoes back the data packets it receives to the sending device.

Device discovery

Network discovery

Use the network discovery command to discover all devices that have joined a network. Issuing the **ND** command sends a broadcast network discovery command throughout the network. All devices that receive the command send a response that includes:

- Device addressing information
- Node identifier string (see NI (Node Identifier))
- Other relevant information

You can use this command for generating a list of all module addresses in a network.

Neighbor polling

Use the neighbor poll command to discover the modules which are immediate neighbors (within RF range) of a particular node. You can use this command to determining network topology and determining possible routes.

The device sends the command using the **FN** command. You can initiate the **FN** command locally on a node using AT command mode or by using a local AT command request frame. You can also initiate the command remotely by sending the target node an **FN** command using a remote AT command request API frame.

A node that executes an **FN** command sends a broadcast to all of its immediate neighbors. All devices that receive this broadcast send an RF packet to the node that initiated the **FN** command. In an instance where the device initiates the command remotely, it sends the responses directly to the node which sent the **FN** command to the target node. The device outputs the response packet on the initiating radio in the same format as a network discovery frame.

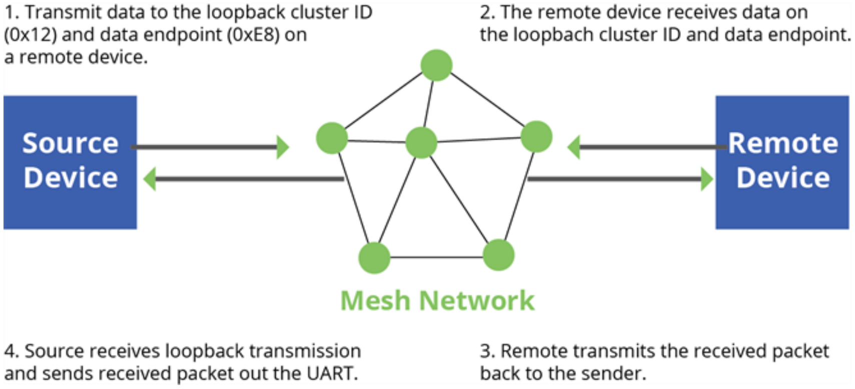
Link reliability

To install a successful mesh network, you must be able to determine where to place individual XBee devices to establish reliable links throughout the mesh network.

Network link testing

To determine the success rate of many transmissions, send unicast data through the network from one device to another to measure the performance of the mesh network.

To simplify link testing, the modules support a loopback cluster ID (0x12) on the data endpoint (0xE8). The device transmits any data sent to this cluster ID on the data endpoint back to the sender as illustrated in the following figure:


1. Transmit data to the loopback cluster ID (0x12) and data endpoint (0xE8) on a remote device.
2. The remote device receives data on the loopbach cluster ID and data endpoint.
Source Device
Remote Device
Mesh Network
4. Source receives loopback transmission and sends received packet out the UART.
3. Remote transmits the received packet back to the sender.

The configuration steps to send data to the loopback cluster ID depend on the AP setting.

AT configuration (AP=0)

To send data to the loopback cluster ID on the data endpoint of a remote device, set the **CI** command value to 0x12. Set the **SE** and **DE** commands set to 0xE8 (default value). Set the **DH** and **DL** commands to the address of the remote. After exiting command mode, the source device transmits any received serial characters to the remote device, and returned to the sender.

API configuration (AP=1 or AP=2)

Send an Explicit Addressing Command API frame (0x11) using 0x12 as the cluster ID and 0xE8 as the source and destination endpoint. The remote device echoes any data packets it receives to the sender.

Link testing between adjacent devices

To test the quality of a link between two adjacent nodes in a network, use the Test Link Request Cluster ID send a number of test packets between any two nodes in a network.

Initiate a link test using an Explicit TX Request frame. Address the command frame to the Test Link Request Cluster ID (0x0014) on destination endpoint 0xE6 on the device to execute the test link. The Explicit TX Request frame contains a 12 byte payload with the following format:

Number of bytes	Field name	Description
8	Destination address	The address the device tests its link with.
2	Payload size	The size of the test packet. Use the MP command to query the maximum payload size for this device.
2	Iterations	The number of packets to send. Use a number between 1 and 4000.

After completing the transmissions of the test link packets, the executing device sends the following data packet to the requesting device's Test Link Result Cluster (0x0094) on endpoint (0xE6). If the requesting device is operating in API mode, the device outputs the following information as an API Explicit RX Indicator Frame:

Number of bytes	Field name	Description
8	Destination address	The address where the device tested its link.
2	Payload size	The size of the test packet sent to test the link.
2	Iterations	The number of packets sent.
2	Success	The number of packets successfully acknowledged.
2	Retries	The total number of MAC retries to transfer all the packets.
1	Result	0x00 - command was successful. 0x03 - invalid parameter used.
1	RR	The maximum number of MAC retries allowed.
1	maxRSSI	The strongest RSSI reading observed during the test.
1	minRSSI	The weakest RSSI reading observed during the test.
1	avgRSSI	The average RSSI reading observed during the test.

Example

Suppose that the link between device A (**SH**/**SL** = 0x0013a20040521234) and device B (**SH**/**SL**=0x0013a2004052abcd) is being tested by transmitting 1,000 40 byte packets. Send the following API packet to the serial interface of the device outputting the results, device C. Note that device C can be the same device as device A or B (Whitespace delineates fields and bold text is the payload portion of the packet):

7E 0020 11 01 0013A20040521234 FFFE E6 E6 0014 C105 00 00 0013A2004052ABCD 0028 03E8 EB

And the following is a possible packet returned:

7E 0027 91 0013A20040521234 FFFE E6 E6 0094 C105 00 0013A2004052ABCD 0028 03E8 03E7 0064 00 0A 50 53 52 9F

(999 out of 1000 packets successful, 100 retries used, RR=10, maxRSSI= **-** 80 dBm, minRSSI= **-** 83 dBm, avgRSSI= **-** 82 dBm)

If the result field is not equal to zero then an error occurred. Ignore the other fields in the packet. If the Success field is equal to zero then ignore the RSSI fields.

Trace routing

Determining the route a DigiMesh unicast takes to its destination is useful when setting up a network or diagnosing problems within a network. Use the Trace Route API option of Tx Request Packets to transmit routing information packets to the originator of a DigiMesh unicast by the intermediate nodes. For a description of the API frames, see API operating mode.

When a unicast is sent with the Trace Route API option enabled, the unicast is sent to its destination radios which forward the unicast to its eventual destination and transmit a Route Information—**RI**—packet back along the route to the unicast originator. For more information, see API operating mode.

Example:

Suppose you unicast a data packet with the trace route enabled from radio A to radio E, through radios B, C, and D. The following sequence occurs:

- After the successful MAC transmission of the data packet from A to B, A outputs an RI Packet indicating that the transmission of the data packet from A to E was successfully forwarded one hop from A to B.
- After the successful MAC transmission of the data packet from B to C, B transmits a RI Packet to A. Then, A outputs this RI packet out its serial interface.
- After the successful MAC transmission of the data packet from C to D, C transmits a RI Packet to A—through B. Then, A outputs this RI packet out its serial interface.
- After the successful MAC transmission of the data packet from D to E, D transmits an RI Packet to A—through C and B. Then, A outputs this RI packet out its serial interface.

Route Information packets are not guaranteed to arrive in the same order as the unicast packet took. It is also possible Route Information packets that are transferred on a weak route to fail before arriving at the unicast originator.

Because of the large number of Route Information packets that can be generated by a unicast with Trace Route enabled, we suggest that the Trace Route option only be used for occasional diagnostic purposes and not for normal operations.

NACK messages

Transmit Request (0x10 and 0x11) frames contain a negative-acknowledge character (NACK) API option (Bit 2 of the Transmit Options field).

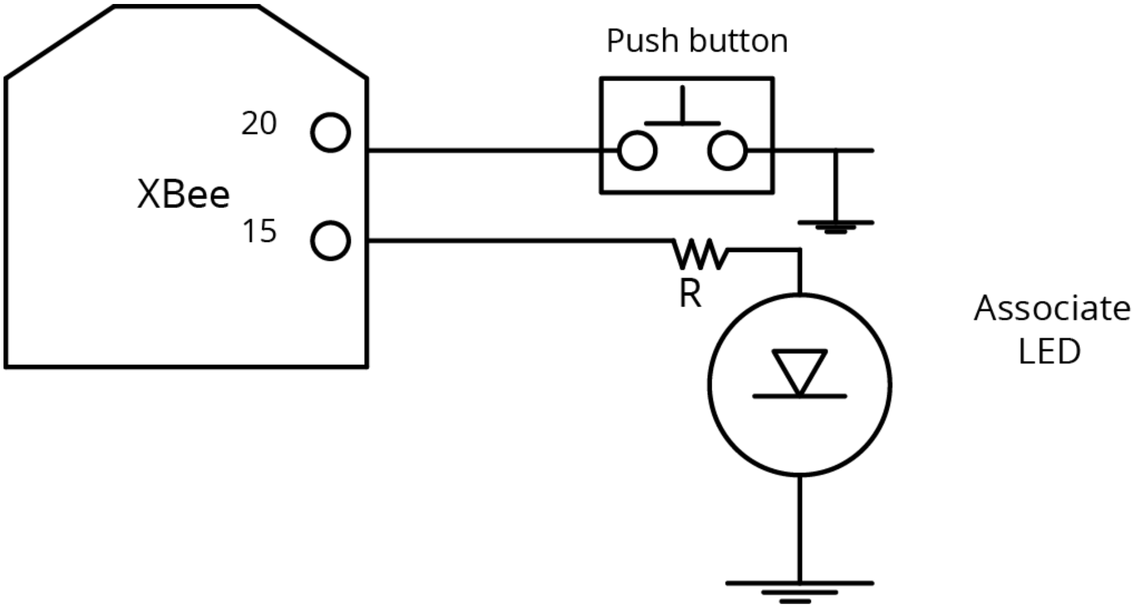
If you use this option when transmitting data, when a MAC acknowledgment failure occurs on one of the hops to the destination device, the device generates a Route Information Packet (0x8D) frame and sends it to the originator of the unicast.

This information is useful because it allows you to identify and repair marginal links.

Commissioning pushbutton and associate LED

XBee devices support a set of commissioning pushbutton and LED behaviors to aid in device deployment and commissioning. These include the commissioning push button definitions and associate LED behaviors. The following features can be supported in hardware:

TH RF Module



A pushbutton and an LED can be connected to the XBee 868LP RF Module pins 33 and 28 (SMT), or pins 20 and 15 (TH) respectively to support the commissioning pushbutton and associate LED functionalities.

Commissioning pushbutton

The commissioning pushbutton definitions provide a variety of simple functions to help with deploying devices in a network. Enable the commissioning button functionality on pin 20 by setting the **D0** command to 1 (enabled by default).

Button Presses	Sleep configuration and sync status	Action
1	Not configured for sleep	Immediately sends a Node Identification broadcast transmission. All devices that receive this transmission blink their Associate LED rapidly for 1 second. All API devices that receive this transmission send a Node Identification frame out their serial interface (API ID 0x95).
1	Configured for synchronous sleep	Wakes the module for 30 seconds. Immediately sends a Node Identification broadcast transmission. All devices that receive this transmission blink their Associate LED rapidly for 1 second. All API devices that receive this transmission send a Node Identification frame out their serial interface (API ID 0x95).
1	Configured for synchronous sleep	Wakes the module for 30 seconds (or until the synchronized network goes to sleep). Queues a

Button Presses	Sleep configuration and sync status	Action
		Node Identification broadcast transmission sent at the beginning of the next network wake cycle. All devices receiving this transmission blink their Associate LEDs rapidly for 1 second. All API devices that receive this transmission will send a Node Identification frame out their serial interface (API ID 0x95).
2	Not configured for synchronous sleep	No effect.
2	Configured for synchronous sleep	Causes a node configured with sleeping router nomination enabled (see the **SO** command in Sleep modes to immediately nominate itself as the network sleep coordinator.
4	Any	Issues an ATRE to restore module parameters to default values.

Use the **CB** command to simulate button presses in the software. Issue a **CB** command with a parameter set to the number of button presses you want executee. For example, sending **CB1** executes the actions 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 interface as an API Node Identification Indicator frame (0x95).

Associate LED

The Associate pin (pin 15) provides an indication of the device's sleep status and diagnostic information. To take advantage of these indications, connect an LED to the Associate pin.

To enable the Associate LED functionality, set the **D5** command to 1; it is enabled by default. If enabled, the Associate pin is configured as an output. This section describes the behavior of the pin.

Use the **LT** command to override the blink rate of the Associate pin. If you set **LT** to 0, the device uses the default blink time: 500 ms for a sleep coordinator, 250 ms otherwise.

The following table describes the Associate LED functionality.

Sleep mode	LED status	Meaning
0	On, blinking	The device has power and is operating properly
1, 4, 5	Off	The device is in a low power mode
1, 4, 5	On, blinking	The device has power, is awake and is operating properly
7	On, solid	The network is asleep, or the device has not synchronized with the network, or has lost synchronization with the network
7, 8	On, slow blinking	The device is acting as the network sleep coordinator and is

Sleep mode	LED status	Meaning
	(500 ms blink time)	operating properly
7, 8	On, fast blinking (250 ms blink time)	The device is properly synchronized with the network
8	Off	The device is in a low power mode
8	On, solid	The device has not synchronized or has lost synchronization with the network

Diagnostics support

The Associate pin works with the Commissioning Pushbutton to provide additional diagnostic behaviors to aid in deploying and testing a network. If you press the Commissioning Pushbutton once, the device transmits a broadcast Node Identification Indicator (0x95) frame at the beginning of the next wake cycle if the device is sleep compatible, or immediately if the device is not sleep compatible. If you enable the Associate LED functionality using the **D5** command, a device that receives this transmission blinks its Associate pin rapidly for one second.

I/O line monitoring

I/O samples

The XBee 868LP RF Module supports both analog input and digital I/O line modes on several configurable pins.

Queried sampling

Pin configuration commands include the following parameters:

Pin command parameter	Description
0	Unmonitored digital input
1	Reserved for pin-specific alternate functionality
2	Analog input (A/D pins) or PWM output (PWM pins)
3	Digital input, monitored
4	Digital output, low
5	Digital output, high
7	Alternate functionality, where applicable

The following table provides the pin configurations when you set the configuration command for a particular pin.

Device pin name	Device pin number	Configuration command
CD / DIO12	4	**P2**
PWM0 / RSSI / DIO10	6	**P0**
PWM1 / DIO11	7	**P1**
$\overline{\text{DTR}}$ / SLEEP_RQ / DIO8	9	**D8**
AD4 / DIO4	11	**D4**
$\overline{\text{CTS}}$/ DIO7	12	**D7**
ON/$\overline{\text{SLEEP}}$/ DIO9	13	**D9**
ASSOC / AD5 / DIO5	15	**D5**
$\overline{\text{RTS}}$ / DIO6	16	**D6**
AD3 / DIO3	17	**D3**
AD2 / DIO2	18	**D2**
AD1 / DIO1	19	**D1**
AD0 / DIO0 / Commissioning Pushbutton	20	**D0**

Use the **PR** command to enable internal pull up/down resistors for each digital input. Use the **PD** command to determine the direction of the internal pull up/down resistor.

Field	Name	Description
1	Sample sets	Number of sample sets in the packet. Always set to 1.
2	Digital channel mask	Indicates which digital I/O lines have sampling enabled. Each bit corresponds to one digital I/O line on the device. 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 = DTR / SLEEP_RQ / DIO8 bit 9 = ON_SLEEP / DIO9 bit 10 = RSSI/DIO10 bit 11 = PWM/DIO11 bit 12 = CD/DIO12 For example, a digital channel mask of 0x002F means DIO0,1,2,3, and 5 are enabled as digital I/O.
1	Analog channel	Indicates which lines have analog inputs enabled for sampling. Each bit in the analog channel mask corresponds to one analog input channel.

Field	Name	Description
	mask	bit 0 = AD0/DIO0 bit 1 = AD1/DIO1 bit 2 = AD2/DIO2 bit 3 = AD3/DIO3 bit 4 = AD4/DIO4 bit 5 = ASSOC/AD5/DIO5
Variable	Sampled data set	If you enable any digital I/O lines, the first two bytes of the data set indicate the state of all enabled digital I/O. Only digital channels that you enable in the Digital channel mask bytes have any meaning in the sample set. If do not enable any digital I/O on the device, it omits these two bytes. Following the digital I/O data (if there is any), each enabled analog channel returns two bytes. The data starts with AIN0 and continues sequentially for each enabled analog input channel up to AIN5.

If you issue the **IS** command using an Command mode, the device returns a carriage return delimited list containing the fields in the previous list. If you issue a command via an API frame, the device returns an AT command response API frame with the I/O data included in the command data portion of the packet.

Example	Sample AT response
0x01	[1 sample set]
0x0C0C	[Digital Inputs: DIO 2, 3, 10, 11 enabled]
0x03	[Analog Inputs: A/D 0, 1 enabled]
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 a device to take an I/O sample and transmit it to a remote device at a periodic rate. Use the **IR** command to set the periodic sample rate.

- To disable periodic sampling, set **IR** to **0**.
- For all other **IR** values, the firmware samples data when **IR** milliseconds elapse and the sample data transmits to a remote device.

The **DH** and **DL** commands determine the destination address of the I/O samples.

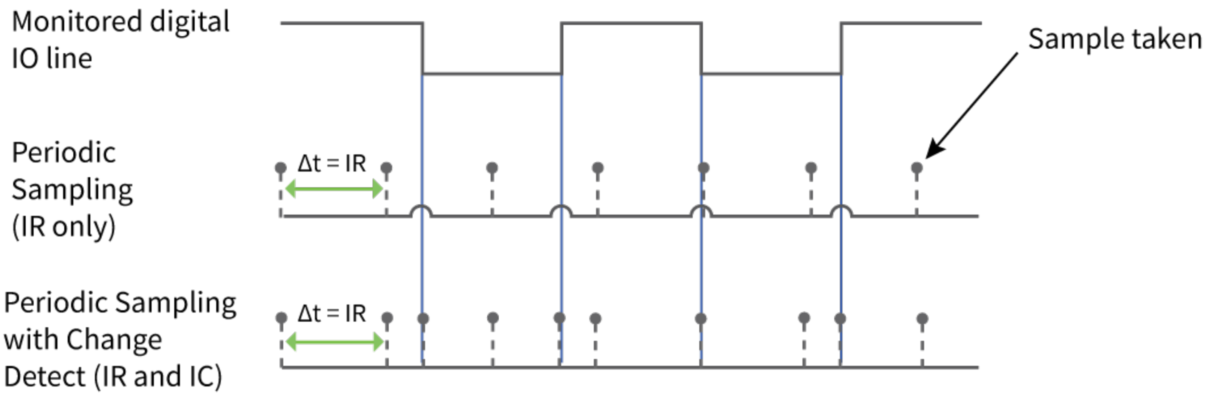
Only devices with API operating mode enabled send I/O data samples out their serial interface. Devices that are in Transparent mode (**AP** = **0**) discard the I/O data samples they receive.

A device with sleep enabled transmits periodic I/O samples at the **IR** rate until the **ST** time expires and the device can resume sleeping. For more information about setting sleep modes, see Sleep modes.

Detect digital I/O changes

You can configure devices to transmit a data sample immediately whenever a monitored digital I/O pin changes state. The **IC** command is a bitmask that you use to set which digital I/O lines to monitor for a state change. If you set one or more bits in **IC**, the device transmits an I/O sample as soon it observes a state change in one of the monitored digital I/O lines using edge detection.

The figure below shows how I/O change detection can work with periodic sampling.


Monitored digital IO line
Sample taken
Periodic Sampling (IR only)
Δt = IR
Periodic Sampling with Change Detect (IR and IC)
Δt = IR

Enabling edge detection forces an immediate sample of all monitored digital I/O lines if any digital I/O lines change state.

General Purpose Flash Memory

XBee 868LP RF Modules provides 119 512-byte blocks of flash memory that an application can read and write to. This memory provides a non-volatile data storage area that an application uses for many purposes. Some common uses of this data storage include:

- Storing logged sensor data
- Buffering firmware update data for a host microcontroller
- Storing and retrieving data tables needed for calculations performed by a host microcontroller

The General Purpose Memory (GPM) is also used to store a firmware update file for over-the-air firmware updates of the device itself.

Access General Purpose Flash Memory

To access the GPM of a target node locally or over-the-air, send commands to the MEMORY_ACCESS cluster ID (0x23) on the DIGI_DEVICE endpoint (0xE6) of the target node using explicit API frames. For a description of Explicit API frames, see Operate in API mode.

To issue a GPM command, format the payload of an explicit API frame as follows:

Byte offset in payload	Number of bytes	Field name	General field description
0	1	GPM_CMD_ID	Specific GPM commands are described in detail in the topics that follow.

Byte offset in payload	Number of bytes	Field name	General field description
1	1	GPM_OPTIONS	Command-specific options.
2	2*	GPM_BLOCK_NUM	The block number addressed in the GPM.
4	2*	GPM_START_INDEX	The byte index within the addressed GPM block.
6	2*	GPM_NUM_BYTES	The number of bytes in the GPM_ DATA field, or in the case of a READ, the number of bytes requested.
8	varies	GPM_DATA	
* Specify multi-byte parameters with big-endian byte ordering.			

When a device sends a GPM command to another device via a unicast, the receiving device sends a unicast response back to the requesting device's source endpoint specified in the request packet. It does not send a response for broadcast requests. If the source endpoint is set to the DIGI_DEVICE endpoint (0xE6) or Explicit API mode is enabled on the requesting device, then the requesting node outputs a GPM response as an explicit API RX indicator frame (assuming it has API mode enabled).

The format of the response is similar to the request packet:

Byte offset in payload	Number of bytes	Field name	General field description
0	1	GPM_CMD_ID	This field is the same as the request field.
1	1	GPM_STATUS	Status indicating whether the command was successful.
2	2*	GPM_BLOCK_NUM	The block number addressed in the GPM.
4	2*	GPM_START_INDEX	The byte index within the addressed GPM block.
6	2*	GPM_NUM_BYTES	The number of bytes in the GPM_DATA field.
8	varies	GPM_DATA	
* Specify multi-byte parameters with big-endian byte ordering.			

Work with flash memory

When working with the General Purpose Memory, observe the following limitations:

- Flash memory write operations are only capable of changing binary 1s to binary 0s. Only the erase operation can change binary 0s to binary 1s. For this reason, you should erase a flash block before performing a write operation.

- When performing an erase operation, you must erase the entire flash memory block—you cannot erase parts of a flash memory block.
- Flash memory has a limited lifetime. The flash memory on which the GPM is based is rated at 20,000 erase cycles before failure. Take care to ensure that the frequency of erase/write operations allows for the desired product lifetime. Digi's warranty does not cover products that have exceeded the allowed number of erase cycles.
- Over-the-air firmware upgrades erase the entire GPM. Any user data stored in the GPM will be lost during an over-the-air upgrade.

General Purpose Flash Memory commands

This section provides information about commands that interact with GPM:

PLATFORM_INFO_REQUEST (0x00)

A PLATFORM_INFO_REQUEST frame can be sent to query details of the GPM structure.

Field name	Command-specific description
GPM_CMD_ID	Should be set to PLATFORM_INFO_REQUEST (0x00).
GPM_OPTIONS	This field is unused for this command. Set to 0.
GPM_BLOCK_NUM	This field is unused for this command. Set to 0.
GPM_START_INDEX	This field is unused for this command. Set to 0.
GPM_NUM_BYTES	This field is unused for this command. Set to 0.
GPM_DATA	No data bytes should be specified for this command.

PLATFORM_INFO (0x80)

When a PLATFORM_INFO_REQUEST command request has been unicast to a node, that node sends a response in the following format to the source endpoint specified in the requesting frame.

Field name	Command-specific description
GPM_CMD_ID	Should be set to PLATFORM_INFO (0x80).
GPM_STATUS	A 1 in the least significant bit indicates an error occurred. All other bits are reserved at this time.
GPM_BLOCK_NUM	Indicates the number of GPM blocks available.
GPM_START_INDEX	Indicates the size, in bytes, of a GPM block.
GPM_NUM_BYTES	The number of bytes in the GPM_DATA field. For this command, this field will be set to 0.
GPM_DATA	No data bytes are specified for this command.

Example

A PLATFORM_INFO_REQUEST sent to a device with a serial number of 0x0013a200407402AC should be formatted as follows (spaces added to delineate fields):

```
7E 001C 11 01 0013A200407402AC FFFE E6 E6 0023 C105 00 00 00 00 0000 0000 0000 24
```

Assuming all transmissions were successful, the following API packets would be output the source node's serial interface:

```
7E 0007 8B 01 FFFE 00 00 00 76
7E 001A 91 0013A200407402AC FFFE E6 E6 0023 C105 C1 80 00 0077 0200 0000 EB
```

ERASE (0x01)

The ERASE command erases (writes all bits to binary 1) one or all of the GPM flash blocks. You can also use the ERASE command to erase all blocks of the GPM by setting the GPM_NUM_BYTES field to 0.

Field name	Command-specific description
GPM_CMD_ID	Should be set to ERASE (0x01).
GPM_OPTIONS	There are currently no options defined for the ERASE command. Set this field to 0.
GPM_BLOCK_NUM	Set to the index of the GPM block that should be erased. When erasing all GPM blocks, this field is ignored (set to 0).
GPM_START_INDEX	The ERASE command only works on complete GPM blocks. The command cannot be used to erase part of a GPM block. For this reason GPM_START_INDEX is unused (set to 0).
GPM_NUM_BYTES	Setting GPM_NUM_BYTES to 0 has a special meaning. It indicates that every flash block in the GPM should be erased (not just the one specified with GPM_BLOCK_NUM). In all other cases, the GPM_NUM_BYTES field should be set to the GPM flash block size.
GPM_DATA	No data bytes are specified for this command.

ERASE_RESPONSE (0x81)

When an ERASE command request has been unicast to a node, that node sends a response in the following format to the source endpoint specified in the requesting frame.

Field name	Command-specific description
GPM_CMD_ID	Should be set to ERASE_RESPONSE (0x81).
GPM_STATUS	A 1 in the least significant bit indicates an error occurred. All other bits are reserved at this time.
GPM_BLOCK_NUM	Matches the parameter passed in the request frame.
GPM_START_INDEX	Matches the parameter passed in the request frame.
GPM_NUM_BYTES	The number of bytes in the GPM_DATA field. For this command,

Field name	Command-specific description
	this field will be set to 0.
GPM_DATA	No data bytes are specified for this command.

Example

To erase flash block 42 of a target radio with serial number of 0x0013a200407402ac format an ERASE packet as follows (spaces added to delineate fields):

```
7E 001C 11 01 0013A200407402AC FFFE E6 E6 0023 C105 00 C0 01 00 002A 0000 0200 37
```

Assuming all transmissions were successful, the following API packets would be output the source node's serial interface:

```
7E 0007 8B 01 FFFE 00 00 00 76
7E 001A 91 0013A200407402AC FFFE E6 E6 0023 C105 C1 81 00 002A 0000 0000 39
```

WRITE (0x02) and ERASE_THEN_WRITE (0x03)

The WRITE command writes the specified bytes to the GPM location specified. Before writing bytes to a GPM block it is important that the bytes have been erased previously. The ERASE_THEN_WRITE command performs an ERASE of the entire GPM block specified with the GPM_BLOCK_NUM field prior to doing a WRITE.

Field name	Command-specific description
GPM_CMD_ID	Should be set to WRITE (0x02) or ERASE_THEN_WRITE (0x03).
GPM_OPTIONS	There are currently no options defined for this command. Set this field to 0.
GPM_BLOCK_NUM	Set to the index of the GPM block that should be written.
GPM_START_INDEX	Set to the byte index within the GPM block where the given data should be written.
GPM_NUM_BYTES	Set to the number of bytes specified in the GPM_DATA field. Only one GPM block can be operated on per command. For this reason, GPM_START_INDEX + GPM_NUM_BYTES cannot be greater than the GPM block size. The number of bytes sent in an explicit API frame (including the GPM command fields) cannot exceed the maximum payload size of the device. The maximum payload size can be queried with the **NP** command.
GPM_DATA	The data to be written.

WRITE _RESPONSE (0x82) and ERASE_THEN_WRITE_RESPONSE (0x83)

When a WRITE or ERASE_THEN_WRITE command request has been unicast to a node, that node sends a response in the following format to the source endpoint specified in the requesting frame.

Field name	Command-specific description
GPM_CMD_ID	Should be set to WRITE_RESPONSE (0x82) or ERASE_THEN_WRITE_ RESPONSE (0x83)
GPM_STATUS	A 1 in the least significant bit indicates an error occurred. All other bits are reserved at this time
GPM_BLOCK_NUM	Matches the parameter passed in the request frame
GPM_START_INDEX	Matches the parameter passed in the request frame
GPM_NUM_BYTES	The number of bytes in the GPM_DATA field. For this command, this field will be set to 0
GPM_DATA	No data bytes are specified for these commands

Example

To write 15 bytes of incrementing data to flash block 22 of a target radio with serial number of 0x0013a200407402ac a WRITE packet should be formatted as follows (spaces added to delineate fields):

```
7E 002B 11 01 0013A200407402AC FFFE E6 E6 0023 C105 00 C0 02 00 0016 0000 000F
0102030405060708090A0B0C0D0E0F C5
```

Assuming all transmissions were successful and that flash block 22 was previously erased, the following API packets would be output the source node's serial interface:

```
7E 0007 8B 01 FFFE 00 00 00 76
7E 001A 91 0013A200407402AC FFFE E6 E6 0023 C105 C1 82 00 0016 0000 0000 4C
```

READ (0x04)

You can use the READ command to read the specified number of bytes from the GPM location specified. Data can be queried from only one GPM block per command.

Field name	Command-specific description
GPM_CMD_ID	Should be set to READ (0x04).
GPM_OPTIONS	There are currently no options defined for this command. Set this field to 0.
GPM_BLOCK_NUM	Set to the index of the GPM block that should be read.
GPM_START_INDEX	Set to the byte index within the GPM block where the given data should be read.
GPM_NUM_BYTES	Set to the number of data bytes to be read. Only one GPM block can be operated on per command. For this reason, GPM_START_INDEX + GPM_NUM_BYTES cannot be greater than the GPM block size. The number of bytes sent in an explicit API frame (including the GPM command fields) cannot exceed the maximum payload size of the device. You can query the maximum payload size with the **NP** AT command.
GPM_DATA	No data bytes should be specified for this command.

READ_RESPONSE (0x84)

When a READ command request has been unicast to a node, that node sends a response in the following format to the source endpoint specified in the requesting frame.

Field name	Command-specific description
GPM_CMD_ID	Should be set to READ_RESPONSE (0x84).
GPM_STATUS	A 1 in the least significant bit indicates an error occurred. All other bits are reserved at this time.
GPM_BLOCK_NUM	Matches the parameter passed in the request frame.
GPM_START_INDEX	Matches the parameter passed in the request frame.
GPM_NUM_BYTES	The number of bytes in the GPM_DATA field.
GPM_DATA	The bytes read from the GPM block specified.

Example

To read 15 bytes of previously written data from flash block 22 of a target radio with serial number of 0x0013a200407402ac a READ packet should be formatted as follows (spaces added to delineate fields):

```
7E 001C 11 01 0013A200407402AC FFFE E6 E6 0023 C105 00 C0 04 00 0016 0000 000F 3B
```

Assuming all transmissions were successful and that flash block 22 was previously written with incrementing data, the following API packets would be output the source node's serial interface:

```
7E 0007 8B 01 FFFE 00 00 00 76
7E 0029 91 0013A200407402AC FFFE E6 E6 0023 C105 C1 84 00 0016 0000 000F
0102030405060708090A0B0C0D0E0F C3
```

FIRMWARE_VERIFY (0x05) and FIRMWARE_VERIFY_AND_INSTALL (0x06)

Use the FIRMWARE_VERIFY and FIRMWARE_VERIFY_AND_INSTALL commands when remotely updating firmware on a device. For more information about firmware updates. These commands check if the GPM contains a valid over-the-air update file. For the FIRMWARE_VERIFY_AND_INSTALL command, if the GPM contains a valid firmware image then the device resets and begins using the new firmware.

Field name	Command-specific description
GPM_CMD_ID	Should be set to FIRMWARE_VERIFY (0x05) or FIRMWARE_VERIFY_AND_INSTALL (0x06)
GPM_OPTIONS	There are currently no options defined for this command. Set this field to 0.
GPM_BLOCK_NUM	This field is unused for this command. Set to 0.
GPM_START_INDEX	This field is unused for this command. Set to 0.
GPM_NUM_BYTES	This field is unused for this command. Set to 0.
GPM_DATA	This field is unused for this command

FIRMWARE_VERIFY_RESPONSE (0x85)

When a FIRMWARE_VERIFY command request has been unicast to a node, that node sends a response in the following format to the source endpoint specified in the requesting frame.

Field name	Command-specific description
GPM_CMD_ID	Should be set to FIRMWARE_VERIFY_RESPONSE (0x85)
GPM_STATUS	A 1 in the least significant bit indicates the GPM does not contain a valid firmware image. A 0 in the least significant bit indicates the GPM does contain a valid firmware image. All other bits are reserved at this time.
GPM_BLOCK_NUM	This field is unused for this command. Set to 0.
GPM_START_INDEX	This field is unused for this command. Set to 0.
GPM_NUM_BYTES	This field is unused for this command. Set to 0.
GPM_DATA	This field is unused for this command

FIRMWARE_VERIFY _AND_INSTALL_RESPONSE (0x86)

When a FIRMWARE_VERIFY_AND_INSTALL command request has been unicast to a node, that node sends a response in the following format to the source endpoint specified in the requesting frame only if the GPM memory does not contain a valid image. If the image is valid, the device resets and begins using the new firmware.

Field name	Command-specific description
GPM_CMD_ID	Should be set to FIRMWARE_VERIFY_AND_INSTALL_RESPONSE (0x86).
GPM_STATUS	A 1 in the least significant bit indicates the GPM does not contain a valid firmware image. All other bits are reserved at this time.
GPM_BLOCK_NUM	This field is unused for this command. Set to 0.
GPM_START_INDEX	This field is unused for this command. Set to 0.
GPM_NUM_BYTES	This field is unused for this command. Set to 0.
GPM_DATA	This field is unused for this command.

Example

To verify a firmware image previously loaded into the GPM on a target device with serial number 0x0013a200407402ac, format a FIRMWARE_VERIFY packet as follows (spaces added to delineate fields):

```
7E 001C 11 01 0013A200407402AC FFFE E6 E6 0023 C105 00 00 05 00 0000 0000 0000 1F
```

Assuming all transmissions were successful and that the firmware image previously loaded into the GPM is valid, the following API packets would be output the source node's serial interface:

```
7E 0007 8B 01 FFFE 00 00 00 76
7E 001A 91 0013A200407402AC FFFE E6 E6 0023 C105 C1 85 00 0000 0000 0000 5F
```

Over-the-air firmware updates

There are two methods of updating the firmware on the device. You can update the firmware locally with XCTU using the device's serial port interface. You can also update firmware using the device's RF interface (over-the-air updating.)

The over-the-air firmware update method provided is a robust and versatile technique that you can tailor to many different networks and applications. OTA updates are reliable and minimize disruption of normal network operations.

In the following sections, we refer to the node that will be updated as the target node. We refer to the node providing the update information as the source node. In most applications the source node is locally attached to a computer running update software.

There are three phases of the over-the-air update process:

1. Distribute the new application
2. Verify the new application
3. Install the application

Distribute the new application

The first phase of performing an over-the-air update on a device is transferring the new firmware file to the target node. Load the new firmware image in the target node's GPM prior to installation. XBee 868LP RF Modules use an encrypted binary (.ebin) file for both serial and over-the-air firmware updates. These firmware files are available on the Digi Support website and via XCTU.

Send the contents of the .ebin file to the target device using general purpose memory WRITE commands. Erase the entire GPM prior to beginning an upload of an .ebin file. The contents of the .ebin file should be stored in order in the appropriate GPM memory blocks. The number of bytes that are sent in an individual GPM WRITE frame is flexible and can be catered to the user application.

Example

The example firmware version has an .ebin file of 55,141 bytes in length. Based on network traffic, we determine that sending a 128 byte packet every 30 seconds minimizes network disruption. For this reason, you would divide and address the .ebin as follows:

GPM_BLOCK_NUM	GPM_START_INDEX	GPM_NUM_BYTES	.ebin bytes
0	0	128	0 to 127
0	128	128	128 to 255
0	256	128	256 to 383
0	384	128	384 to 511
1	0	128	512 to 639
1	128	128	640 to 767
-	-	-	-
-	-	-	-
-	-	-	-

GPM_BLOCK_NUM	GPM_START_INDEX	GPM_NUM_BYTES	.ebin bytes
107	0		54784 to 54911
107	128		54912 to 55039
107	256	101	55040 to 55140

Verify the new application

For an uploaded application to function correctly, every single byte from the .ebin file must be properly transferred to the GPM. To guarantee that this is the case, GPM VERIFY functions exist to ensure that all bytes are properly in place. The FIRMWARE_VERIFY function reports whether or not the uploaded data is valid. The FIRMWARE_VERIFY_AND_INSTALL command reports if the uploaded data is invalid. If the data is valid, it begins installing the application. No installation takes place on invalid data.

Install the application

When the entire .ebin file is uploaded to the GPM of the target node, you can issue a FIRMWARE_VERIFY_AND_INSTALL command. Once the target receives the command it verifies the .ebin file loaded in the GPM. If it is valid, then the device installs the new firmware. This installation process can take up to eight seconds. During the installation the device is unresponsive to both serial and RF communication. To complete the installation, the target module resets. AT parameter settings which have not been written to flash using the **WR** command will be lost.

Important considerations

The firmware upgrade process requires that the device resets itself. Write all parameters with the **WR** command before performing a firmware update. Packet routing information is also lost after a reset. Route discoveries are necessary for DigiMesh unicasts involving the updated node as a source, destination, or intermediate node.

Because explicit API Tx frames can be addressed to a local node (accessible via the SPI or UART) or a remote node (accessible over the RF port) the same process can be used to update firmware on a device in either case.

Networking methods

This section explains the basic layers and the three networking methods available on the XBee 868LP RF Modules, building from the simplest to the most complex.

Directed Broadcast/Repeater mode

In this delivery method, the device sends all outgoing transmissions as broadcasts. Unicast messages are sent as broadcasts, but are addressed to a specific receiver. Only the specified device will emit the received frame out of the serial port.

- Directed broadcast over hops is only available when operating at the 80k RF data rate (**BR** = **1**). The 10k data rate will only address adjacent devices and does not repeat received packets.
- By default the **CE** parameter is set to route all broadcasts. As such, all nodes that receive a repeated packet will repeat it. If you change the **CE** parameter, you can limit which nodes repeat packets, which helps dense networks from becoming overly congested while packets are being repeated.

Point to Point/Multipoint mode

In this mode, there is a permanent link between two endpoints. Switched point-to-point topologies are the basic model of conventional telephony. The value of a permanent point-to-point network is unimpeded communications between the two endpoints. The value of an on-demand point-to-point connection is proportional to the number of potential pairs of subscribers.

Permanent (dedicated)

One of the variations of point-to-point topology is a point-to-point communications channel that appears, to the user, to be permanently associated with the two endpoints. Within many switched telecommunications systems, it is possible to establish a permanent circuit. One example might be a telephone in the lobby of a public building that is programmed to ring only the number of a telephone dispatcher. “Nailing down” a switched connection saves the cost of running a physical circuit between the two points. The resources in such a connection can be released when it is no longer needed.

Switched

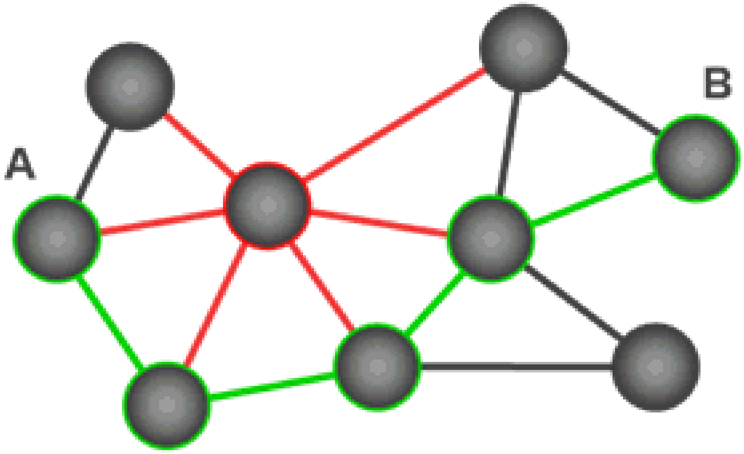
Using circuit-switching or packet-switching technologies, you can set up a point-to-point circuit dynamically and dropped when no longer needed.

DigiMesh networking

A mesh network is a topology in which each node in the network is connected to other nodes around it. Each node cooperates in transmitting information. Mesh networking provides these important benefits:

- **Routing**. With this technique, the message is propagated along a path by hopping from node to node until it reaches its final destination.
- **Ad-hoc network creation**. This is an automated process that creates an entire network of nodes on the fly, without any human intervention.
- **Self-healing**. This process automatically figures out if one or more nodes on the network is missing and reconfigures the network to repair any broken routes.
- **Peer-to-peer architecture**. No hierarchy and no parent-child relationships are needed.
- **Quiet protocol**. Routing overhead will be reduced by using a reactive protocol similar to AODV.

- **Route discovery**. Rather than maintaining a network map, routes will be discovered and created only when needed.
- **Selective acknowledgments**. Only the destination node will reply to route requests.
- **Reliable delivery**. Reliable delivery of data is accomplished by means of acknowledgments.
- **Sleep modes.** Low power sleep modes with synchronized wake are supported with variable sleep and wake times.



With mesh networking, the distance between two nodes does not matter as long as there are enough nodes in between to pass the message along. When one node wants to communicate with another, the network automatically calculates the best path.

A mesh network is also reliable and offers redundancy. For example, If a node can no longer operate because it has been removed from the network or because a barrier blocks its ability to communicate, the rest of the nodes can still communicate with each other, either directly or through intermediate nodes.

Note Mesh networks use more bandwidth for administration and therefore have less available for payloads.

DigiMesh feature set

DigiMesh contains the following features:

- **Self-healing**
  Any node may enter or leave the network at any time without causing the network as a whole to fail.
- **Peer-to-peer architecture**
  No hierarchy and no parent-child relationships are needed.
- **Quiet protocol**
  Routing overhead will be reduced by using a reactive protocol similar to AODV.
- **Route discovery**
  Rather than maintaining a network map, routes will be discovered and created only when needed.

- **Selective acknowledgments**
  Only the destination node will reply to route requests.
- **Reliable delivery**
  Reliable delivery of data is accomplished by means of acknowledgments.
- **Sleep modes**
  Low power sleep modes with synchronized wake are supported with variable sleep and wake times.

Networking concepts

This section provides information on configuring DigiMesh devices and network identifiers.

Device Configuration

You can configure DigiMesh devices to act as routers or end devices with the **CE** command. By default, all devices in a DigiMesh network act as routers. Devices configured as routers actively relay network unicast and broadcast traffic.

Network ID

DigiMesh networks are defined with a unique network identifier. Set the identifier using the **ID** command. For devices to communicate they must be configured with the same network identifier. The **ID** parameter allows multiple DigiMesh networks to co-exist on the same physical channel.

Data transmission and routing

This section provides information on data transmission, routing, throughput, and transmission timeouts.

Unicast addressing

When devices transmit using DigiMesh unicast, the network uses retries and acknowledgments (ACKs) for reliable data delivery. In a retry and acknowledgment scheme, for every data packet that a device sends, the receiving device must send an acknowledgment back to the transmitting device to let the sender know that the data packet arrived at the receiver. If the transmitting device does not receive an acknowledgment then it re-sends the packet. It sends the packet a finite number of times before the system times out.

The **MR** (Mesh Network Retries) parameter determines the number of mesh network retries. The sender device transmits RF data packets up to **MR** + 1 times across the network route, and the receiver transmits ACKs when it receives the packet. If the sender does not receive a network ACK within the time it takes for a packet to traverse the network twice, the sender retransmits the packet.

To send unicast messages while in Transparent operating mode, set the **DH** and **DL** on the transmitting device to match the corresponding **SH** and **SL** parameter values on the receiving device.

Broadcast addressing

All of the routers in a network receive and repeat broadcast transmissions. Broadcast transmissions do not use ACKs, so the sending device sends the broadcast multiple times. By default, the sending device sends a broadcast transmission four times. The transmissions become automatic retries without acknowledgments. This results in all nodes repeating the transmission four times as well.

In order to avoid RF packet collisions, the network inserts a random delay before each router relays the broadcast message. You can change this random delay time with the **NN** parameter.

Sending frequent broadcast transmissions can quickly reduce the available network bandwidth. Use broadcast transmissions sparingly.

The broadcast address is a 64 bit address with the lowest 16 bits set to 1. The upper bits are set to 0. To send a broadcast transmission:

- Set **DH** to 0.
- Set **DL** to 0xFFFF.

In API operating mode, this sets the destination address to 0x000000000000FFFF.

Routing

Devices within a mesh network determine reliable routes using a routing algorithm and table. The routing algorithm uses a reactive method derived from Ad-hoc On-demand Distance Vector (AODV). The device uses an associative routing table to map a destination node address with its next hop. By sending a message to the next hop address, the message reaches its destination or is forwarded to an intermediate router which routes the message on to its destination.

The device broadcasts a message with a broadcast address to all neighbors. All routers receiving the message rebroadcast the message **MT**+1 times and eventually the message reaches all corners of the network. Packet tracking prevents a node from resending a broadcast message more than **MT**+1 times.

Route discovery

Route discovery is a process that occurs when:

1. The source node does not have a route to the requested destination.
2. A route fails. This happens when the source node uses up its network retries without receiving an ACK.

Route discovery begins by the source node broadcasting a route request (RREQ). We call any router that receives the RREQ and is not the ultimate destination, an intermediate node.

Intermediate nodes may either drop or forward a RREQ, depending on whether the new RREQ has a better route back to the source node. If so, the node saves, updates and broadcasts the RREQ.

When the ultimate destination receives the RREQ, it unicasts a route reply (RREP) back to the source node along the path of the RREQ. It does this regardless of route quality and regardless of how many times it has seen an RREQ before.

This allows the source node to receive multiple route replies. The source node selects the route with the best round trip route quality, which it uses for the queued packet and for subsequent packets with the same destination address.

DigiMesh throughput

Throughput in a DigiMesh network can vary due to a number of variables, including:

- The number of hops.
- If you enable or disable encryption.
- Sleeping end devices.
- Failures and route discoveries.

The results apply to the 80 kb/s version, 115.2 kb/s serial data rate, 100 KB.

Configuration	Data throughput
Mesh unicast, 1 hop, encryption disabled	35.6 kb/s
Mesh unicast, 3 hop, encryption disabled	11.9 kb/s
Mesh unicast, 6 hop, encryption disabled	7.1 kb/s
Mesh unicast, 1 hop, encryption enabled	35.3 kb/s
Mesh unicast, 3 hop, encryption enabled	11.8 kb/s
Mesh unicast, 6 hop, encryption enabled	7.0 kb/s
Point to point unicast, encryption disabled	54.7 kb/s
Point to point unicast, encryption enabled	53.9 kb/s

Configuration	Data throughput
Point to point unicast, encryption disabled	8.4 kb/s
Point to point unicast, encryption enabled	8.3 kb/s

Note We made the data throughput measurements by setting the serial interface rate to 115200 b/s, and measuring the time to send 100,000 bytes from source to destination. During the test, no route discoveries or failures occurred.

Transmission timeouts

When a device in API operating mode receives a Transmit Request (0x10, 0x11) frame, or a device in Transparent operating mode meets the packetization requirements (**RO**, **RB**), the time required to route the data to its destination depends on:

- A number of configured parameters.
- Whether the transmission is a unicast or a broadcast.
- If the route to the destination address is known.

Timeouts or timing information is provided for the following transmission types:

- Broadcast transmission
- Unicast transmission on a known route
- Unicast transmission on an unknown route
- Unicast transmission on a broken route

Note The timeouts in this documentation are theoretical timeouts and are not precisely accurate. Your application should pad the calculated maximum timeouts by a few hundred milliseconds. When you use API operating mode, use Extended Transmit Status - 0x8B as the primary method to determine if a transmission is complete.

Unicast one hop time

unicastOneHopTime is a building block of many of the following calculations. It represents the amount of time it takes to send a unicast transmission between two adjacent nodes. The amount of time depends on the **%H** parameter.

Transmit a broadcast

All of the routers in a network must relay a broadcast transmission.

The maximum delay occurs when the sender and receiver are on the opposite ends of the network.

The **NH** and **%H** parameters define the maximum broadcast delay as follows:

BroadcastTxTime = NH * NN * %8

Unless **BH** < **NH**, in which case the formula is:

BroadcastTxTime = BH * NN * %8

Transmit a unicast with a known route

When a device knows a route to a destination node, the transmission time is largely a function of the number of hops and retries. The timeout associated with a unicast assumes that the maximum number of hops is necessary, as specified by the **NH** command.

You can estimate the timeout in the following manner:

knownRouteUnicastTime=2*NH*MR*unicastOneHopTime

Transmit a unicast with an unknown route

If the transmitting device does not know the route to the destination, it begins by sending a route discovery. If the route discovery is successful, then the transmitting device transmits data. You can estimate the timeout associated with the entire operation as follows:

unknownRouteUnicastTime=BroadcastTxTime+
(**NH***unicastOneHopTime)+knownRouteUnicastTime

Transmit a unicast with a broken route

If the route to a destination node changes after route discovery completes, a node begins by attempting to send the data along the previous route. After it fails, it initiates route discovery and, when the route discovery finishes, transmits the data along the new route. You can estimate the timeout associated with the entire operation as follows:

brokenRouteUnicastTime=BroadcastTxTime+(NH*unicastOneHopTime)+
(2*knownRouteUnicastTime)

AT commands

Special commands

The following commands are special commands.

AC (Apply Changes)

This command applies to the XBee 868LP RF Module.

Immediately applies new settings without exiting Command mode.

Parameter range

N/A

Default

N/A

FR (Software Reset)

This command applies to the XBee 868LP RF Module.

Resets the device. The device responds immediately with an **OK** and performs a reset 100 ms later.

If you issue **FR** while the device is in Command Mode, the reset effectively exits Command mode.

Parameter range

N/A

Default

N/A

RE (Restore Defaults)

This command applies to the XBee 868LP RF Module.

Restore device parameters to factory defaults.

Parameter range

N/A

Default

N/A

WR (Write

This command applies to the XBee 868LP RF Module.

Writes parameter values to non-volatile memory so that parameter modifications persist through subsequent resets.

Note Once you issue a **WR** command, do not send any additional characters to the device until after you receive the **OK** response.

Parameter range

N/A

Default

N/A

MAC/PHY commands

The following AT commands are MAC/PHY commands.

CM (Channel Mask)

This command applies to the XBee 868LP RF Module.

CM allows you to selectively enable or disable channels used for RF communication. This is useful to avoid using frequencies that experience unacceptable levels of RF interference, or to operate two networks of radios on separate frequencies.

This mask limits the channels where the device transmits. See Technical specifications for the list of frequencies. Channel 0 is bit 0. You must enable at least two channels, except when using only a single frequency of 869.85 MHz. When you use this mode (use 0x20000000), LBT+AFA is disabled and requires the power level to be 5 mW e.r.p. or less.

This command is a bitfield.

Exactly number of channels must be made available for the device to communicate on.

The **CM** command does not limit receive channels. If two devices have mutually exclusive values for **CM** (for example 0x0000FF00 and 0x000000FF), then communication is possible because both devices still listen on all possible channels, while limiting the transmission channels to those specified in the **CM** command.

Parameter range

0 - 0x3FFFFFFF [bitfield]

Default

Europe: 0x3FFFFFFF (channels 0 - 29, 863.15 - 869.85 MHz)

Europe (single frequency mode): 0x20000000 (channel 29, 869.85 MHz)

HP (Preamble ID)

This command applies to the XBee 868LP RF Module.

The preamble ID for which the device communicates. Only devices with matching preamble IDs can communicate with each other. Different preamble IDs minimize interference between multiple sets of devices operating in the same vicinity. When receiving a packet, the device checks this before the network ID, as it is encoded in the preamble, and the network ID is encoded in the MAC header.

Parameter range

0 - 9

Default

0

ID (Network ID)

This command applies to the XBee 868LP RF Module.

Set or read the user network identifier.

Devices must have the same network identifier to communicate with each other.

When receiving a packet, the device check this after the preamble ID. If you are using Original equipment manufacturer (OEM) network IDs, **0xFFFF** uses the factory value.

Parameter range

0 - 0x7FFF

Default

0x7FFF

MT (Broadcast Multi-Transmits)

This command applies to the XBee 868LP RF Module.

Set or read the number of additional MAC-level broadcast transmissions. All broadcast packets are transmitted **MT**+1 times to ensure they are received.

Parameter range

0 - 5

Default

3

PL (TX Power Level)

This command applies to the XBee 868LP RF Module.

Sets or displays the power level at which the device transmits conducted power. Power levels are approximate.

These values include the gain of a 2 dBi antenna. The conducted power is 2 dBi less.

Parameter range

These parameters equate to the following settings for the XBee 868LP RF Module.

Setting	Power level
0	2 mW EIRP
1	5 mW EIRP
2	10 mW EIRP
3	16 mW EIRP
4	25 mW EIRP

Default

4

RR (Unicast Mac Retries)

This command applies to the XBee 868LP RF Module.

Set or read the maximum number of MAC level packet delivery attempts for unicasts. If **RR** is non-zero, the sent unicast packets request an acknowledgment from the recipient. Unicast packets can be retransmitted up to **RR** times if the transmitting device does not receive a successful acknowledgment.

Parameter range

0 - 0xF

Default

0x10

ED (Energy Detect)

This command applies to the XBee 868LP RF Module.

Starts an energy detect scan. This command accepts an argument to specify the time in milliseconds to scan all channels. The device loops through all the available channels until the time elapses. It returns the maximal energy on each channel, a comma follows each value, and the list ends with a carriage return. The values returned reflect the energy level that **ED** detects in -dBm units.

Parameter range

0 - 0xFF

Default

0x10

Diagnostic commands

The following AT commands are diagnostic commands. Diagnostic commands are typically volatile and will not persist across a power cycle.

BC (Bytes Transmitted)

This command applies to the XBee 868LP RF Module.

The number of RF bytes transmitted. The firmware counts every byte of every packet, including MAC/PHY headers and trailers. The purpose of this count is to estimate battery life by tracking time spent performing transmissions.

This number rolls over to **0** from **0xFFFF**.

You can reset the counter to any unsigned 16-bit value by appending a hexadecimal parameter to the command.

Parameter range

0 - 0xFFFF

Default

0

DB (Last Packet RSSI)

This command applies to the XBee 868LP RF Module.

Reports the RSSI in -dBm of the last received RF data packet. **DB** returns a hexadecimal value for the -dBm measurement.

For example, if **DB** returns 0x60, then the RSSI of the last packet received was -96 dBm.

The XBee 868LP RF Module reports RSSI values within approximately 15 dBm of the sensitivity level of the device.

Signals which exceed approximately -85 dBm are reported as approximately -85 dBm.

DB only indicates the signal strength of the last hop. It does not provide an accurate quality measurement for a multihop link.

Parameter range

0 - 0xFF [read-only]

Default

0

ER (Received Error Count)

This command applies to the XBee 868LP RF Module.

This count increments when a device receives a packet that contains integrity errors of some sort. When the number reaches 0xFFFF, the firmware does not count further events.

To reset the counter to any 16-bit unsigned value, append a hexadecimal parameter to the **ER** command.

Parameter range

0 - 0xFFFF

Default

0

GD (Good Packets Received)

This command applies to the XBee 868LP RF Module.

This count increments when a device receives a good frame with a valid MAC header on the RF interface. Once the number reaches 0xFFFF, it does not count further events.

Parameter range

0 - 0xFFFF

Default

0

EA (MAC ACK Timeouts)

This command applies to the XBee 868LP RF Module.

This count increments whenever a MAC ACK timeout occurs on a MAC-level unicast. When the number reaches **0xFFFF**, the firmware does not count further events.

To reset the counter to any 16-bit value, append a hexadecimal parameter to the command.

Parameter range

0 - 0xFFFF

Default

0

TR (Transmission Errors)

This command applies to the XBee 868LP RF Module.

This count increments whenever a MAC transmission attempt exhausts all MAC retries without ever receiving a MAC acknowledgment message from the destination node. Once the number reaches **0xFFFF**, it does not count further events.

To reset the counter to any 16-bit value, append a hexadecimal parameter to the command.

Parameter range

0 - 0xFFFF

Default

0

UA (MAC Unicast Transmission Count)

This command applies to the XBee 868LP RF Module.

This count increments whenever a MAC unicast transmission occurs that requests an ACK. Once the number reaches 0xFFFF, it does not count further events.

You can reset the counter to any 16-bit unsigned value by appending a hexadecimal parameter to the command.

Parameter range

0 - 0xFFFF

Default

0

%H (MAC Unicast One Hop Time)

This command applies to the XBee 868LP RF Module.

The MAC unicast one hop time timeout in milliseconds. If you change the MAC parameters it can change this value.

Parameter range

[read-only]

Default

0xCF

%8 (MAC Broadcast One Hop Time)

This command applies to the XBee 868LP RF Module.

The MAC broadcast one hop time timeout in milliseconds. If you change MAC parameters, it can change this value.

Parameter range

[read-only]

Default

0x1BE

Network commands

The following commands are network commands.

CE (Node Messaging Options)

This command applies to the XBee 868LP RF Module.

The routing and messaging mode bit field of the device.

A routing device repeats broadcasts. Indirect Messaging Coordinators do not transmit point-to-multipoint unicasts until an end device requests them. Setting a device as an end device causes it to regularly send polls to its Indirect Messaging Coordinator. Nodes can also be configured to route, or not route, multi-hop packets.

Bit	Description
Bit 0	Indirect Messaging Coordinator enable. All point-to-multipoint unicasts will be held until requested by a polling end device.
Bit 1	Disable routing on this node. When set, this node will not propagate broadcasts or become an intermediate node in a DigiMesh route. This node will not function as a repeater.
Bit 2	Indirect Messaging Polling enable. Periodically send requests for messages held by the node’s coordinator.

Note Bit 0 and Bit 2 cannot be set at the same time.

Parameter range

0 - 6

Default

0

BH (Broadcast Hops)

This command applies to the XBee 868LP RF Module.

The number of hops for broadcast data transmissions.

Set the value to **0** for the maximum number of hops.

If you set **BH** greater than **NH**, the device uses the value of **NH**.

Parameter range

0 - 0x20

Default

0

NH (Network Hops)

This command applies to the XBee 868LP RF Module.

The maximum number of hops expected to be seen in a network route. This value does not limit the number of hops allowed, but it is used to calculate timeouts waiting for network acknowledgments.

Both variants are supported.

Parameter range

1 - 0x20

Default

7

NN (Network Delay Slots)

This command applies to the XBee 868LP RF Module.

Set or read the maximum random number of network delay slots before rebroadcasting a network packet.

Parameter range

1 - 5

Default

3

MR (Mesh Unicast Retries)

This command applies to the XBee 868LP RF Module.

Set or read the maximum number of network packet delivery attempts. If **MR** is non-zero, the packets a device sends request a network acknowledgment, and can be resent up to **MR**+1 times if the device does not receive an acknowledgment.

We recommend that you set this value to **1**.

If you set this parameter to **0**, it disables network ACKs. Initially, the device can find routes, but a route will never be repaired if it fails.

Parameter range

0 - 7

Default

1

Addressing commands

The following AT commands are addressing commands.

SH (Serial Number High)

This command applies to the XBee 868LP RF Module.

Displays the upper 32 bits of the unique IEEE 64-bit extended address assigned to the XBee in the factory.

Parameter range

0 - 0xFFFFFFFF [read-only]

Default

Set in the factory

SL (Serial Number Low)

This command applies to the XBee 868LP RF Module.

Displays the lower 32 bits of the unique IEEE 64-bit RF extended address assigned to the XBee in the factory.

Parameter range

0 - 0xFFFFFFFF [read-only]

Default

Set in the factory

DH (Destination Address High)

This command applies to the XBee 868LP RF Module.

Set or read the upper 32 bits of the 64-bit destination address. When you combine **DH** with **DL**, it defines the destination address that the device uses for transmissions in Transparent mode.

Parameter range

0 - 0xFFFFFFFF

Default

0

DL (Destination Address Low)

This command applies to the XBee 868LP RF Module.

Set or display the lower 32 bits of the 64-bit destination address. When you combine **DH** with **DL**, it defines the destination address that the device uses for transmissions in Transparent mode.

Parameter range

0 - 0xFFFFFFFF

Default

0x0000FFFF

TO (Transmit Options)

This command applies to the XBee 868LP RF Module.

The bitfield that configures the transmit options for Transparent mode.

The device's transmit options. The device uses these options for all transmissions. You can override these options using the TxOptions field in the API TxRequest frames.

Parameter range

Bit	Meaning	Description
6,7	Delivery method	b’00 = <invalid option> b’01 = Point-multipoint b'10 = Repeater mode—directed broadcast of packets b’11 = DigiMesh—not available on 10k product
5	Reserved	<set this bit to 0>
4	Reserved	<set this bit to 0>
3	Trace Route	Enable a Trace Route on all DigiMesh API packets
2	NACK	Enable a NACK messages on all DigiMesh API packets
1	Disable RD	Disable Route Discovery on all DigiMesh unicasts
0	Disable ACK	Disable acknowledgments on all unicasts

Example 1: Set **TO** to **0x80** to send all transmissions using repeater mode.

Example 2: Set **TO** to **0xC1** to send transmissions using DigiMesh, with network acknowledgments disabled.

- Bits 6 and 7 cannot be set to DigiMesh on the 10k build.
- Bits 4 and 5 must be set to 0.
- Bits 1, 2, and 3 cannot be set on the 10k build.

When you set **BR** to **0** the **TO** option has the DigiMesh and Repeater mode disabled automatically.

Default

0x40 (10k product)

0xC0 (80k product)

NI (Node Identifier)

This command applies to the XBee 868LP RF Module.

Stores the node identifier string for a device, which is a user-defined name or description of the device. This can be up to 20 ASCII characters.

- The command automatically ends when the maximum bytes for the string have been entered.

Use the **ND** (Network Discovery) command with this string as an argument to easily identify devices on the network.

The **DN** command also uses this identifier.

Parameter range

A string of case-sensitive ASCII printable characters from 0 to 20 bytes in length. A carriage return or a comma automatically ends the command.

Default

One ASCII space character (0x20)

NT (Node Discover Timeout)

This command applies to the XBee 868LP RF Module.

Sets the amount of time a base node waits for responses from other nodes when using the ND (Network Discover), DN (Discover Node), and FN (Find Neighbors) commands. When a discovery is performed, the broadcast transmission includes the **NT** value to provide all remote devices with a response timeout. Remote devices wait a random time, less than **NT**, before sending their response to avoid collisions.

The **N?** command should be used to determine how long the actual discovery timeout will be based on current device configuration.

Parameter range

0x20 - 0x2EE0 (x 100 ms)

Default

0x82 (13 seconds)

NO (Node Discovery Options)

This command applies to the XBee 868LP RF Module.

Set or read the network discovery options value for the **ND** (Network Discovery) command on a particular device. The options bit field value changes the behavior of the **ND** command and what optional values the local device returns when it receives an **ND** command or API Node Identification Indicator (0x95) frame.

Parameter range

0x0 - 0x7 (bit field)

Option	Description
0x01	Append the **DD** (Digi Device Identifier) value to **ND** responses or API node identification frames.
0x02	Local device sends **ND** or **FN** (Find Neighbors) response frame when the **ND** is issued.
0x04	Append the RSSI of the last hop to **ND**, **FN**, and responses or API node identification frames.

Default

0x0

CI (Cluster ID)

This command applies to the XBee 868LP RF Module.

The application layer cluster ID value. The device uses this value as the cluster ID for all data transmissions. The default value 0x11 (Transparent data cluster ID).

Parameter range

0 - 0xFFFF

Default

0x11

DE (Destination Endpoint)

This command applies to the XBee 868LP RF Module.

Sets or displays the application layer destination ID value. The value is used as the destination endpoint for all data transmissions. The default value (0xE8) is the Digi data endpoint.

Parameter range

0 - 0xFF

Default

0xE8

SE (Source Endpoint)

This command applies to the XBee 868LP RF Module.

Sets or displays the application layer source endpoint value. The value is used as the source endpoint for all data transmissions. The default value (0xE8) is the Digi data endpoint.

This command only affects outgoing transmissions in transparent mode (**AP** = **0**).

0xE8 is the Digi data endpoint used for outgoing data transmissions.

0xE6 is the Digi device object endpoint used for configuration and commands.

Parameter range

0 - 0xFF

Default

0xE8

Addressing discovery/configuration commands

AG (Aggregator Support)

This command applies to the XBee 868LP RF Module.

The **AG** command sends a broadcast through the network that has the following effects on nodes that receive the broadcast:

- The receiving node establishes a DigiMesh route back to the originating node, if there is space in the routing table.
- The **DH** and **DL** of the receiving node update to the address of the originating node if the **AG** parameter matches the current **DH**/**DL** of the receiving node.
- API-enabled devices with updated **DH** and **DL** send an Aggregate Addressing Update frame (0x8E) out the serial port.

Parameter range

Any 64-bit address

Default

N/A

DN (Discover Node)

This command applies to the XBee 868LP RF Module.

Resolves an **NI** (Node identifier) string to a physical address (case sensitive).

The following events occur after **DN** discovers the destination node:

When **DN** is sent in Command mode:

1. The device sets **DL** and **DH** to the extended (64-bit) address of the device with the matching **NI** string.
2. The receiving device returns **OK** (or **ERROR**).
3. The device exits Command mode to allow for immediate communication. If an **ERROR** is received, the device does not exit Command mode.

When **DN** is sent as an API frame, the receiving device returns 0xFFFE followed by its 64-bit extended addresses in an API Command Response frame.

Parameter range

20-byte ASCII string

Default

N/A

ND (Network Discover)

This command applies to the XBee 868LP RF Module.

Discovers and reports all devices found in the network after a jittered time delay. For each discovered device, the following information is returned:

MY<CR> (2 bytes) (always 0xFFFE)
SH<CR> (4 bytes)
SL<CR> (4 bytes)
DB<CR> (Contains the detected signal strength of the response in negative dBm units)
NI <CR> (variable, 0-20 bytes plus 0x00 character)
PARENT_NETWORK ADDRESS<CR> (2 bytes)
DEVICE_TYPE<CR> (1 byte: **0** = Coordinator, **1** = Router, **2** = End Device)
STATUS<CR> (1 byte: reserved)

PROFILE_ID<CR> (2 bytes)

MANUFACTURER_ID<CR> (2 bytes)

DIGI DEVICE TYPE<CR> (4 bytes. Optionally included based on **NO** settings.)

RSSI OF LAST HOP<CR> (1 byte. Optionally included based on **NO** settings.)

After (**NT** * 100) milliseconds, the command ends by returning a <CR>. **ND** also accepts a NI (Node Identifier) as a parameter (optional). In this case, only a device that matches the supplied identifier responds.

If you send **ND** through a local API frame, the device returns each response as a separate AT_CMD_ Response packet. The data consists of the above listed bytes without the carriage return delimiters. The **NI** string end in a **0x00** null character.

Parameter range

N/A

Default

N/A

FN (Find Neighbors)

This command applies to the XBee 868LP RF Module.

If you send the **FN** command through a local API frame, each response returns as a separate Local or Remote AT Command Response API packet, respectively. The data consists of the bytes in the previous list without the carriage return delimiters. The **NI** string ends in a 0x00 null character.

Parameter range

N/A

Default

N/A

Diagnostic - addressing commands

The following AT command is a Diagnostic - addressing command.

N? (Network Discovery Timeout)

This command applies to the XBee 868LP RF Module.

The maximum response time, in milliseconds, for **ND** (Network Discovery) responses and **DN** (Discover Node) responses. The timeout is the sum of **NT** (Network Discovery Back-off Time) and the network propagation time.

Parameter range

[read-only]

Default

N/A

Security commands

The following AT commands are security commands.

EE (Security Enable)

This command applies to the XBee 868LP RF Module.

Enables or disables Advanced Encryption Standard (AES) encryption.

Set this command parameter the same on all devices in a network.

Parameter range

0 - 1

Parameter	Description
0	Encryption Disabled
1	Encryption Enabled

Default

0

KY (AES Encryption Key)

This command applies to the XBee 868LP RF Module.

Sets the network security key value that the device uses for encryption and decryption.

This command is write-only. If you attempt to read **KY**, the device returns an **OK** status.

Set this command parameter the same on all devices in a network.

The value passes in as hex characters when you set it from AT command mode, and as binary bytes when you set it in API mode.

Parameter range

128-bit value

Default

N/A

Serial interfacing commands

BD (Baud Rate)

This command applies to the XBee 868LP RF Module.

Sets or displays the serial interface baud rate for communication between the device's serial port and the host.

Values from 0 - 8 select preset standard rates.

Values at 0x100 and above select the actual baud rate if the host supports it.

Parameter range

Standard baud rates: 0x0 - 0x8

Non-standard baud rates: 0x100 to 0x6ACFC0

Parameter	Description
0x0	1200 b/s
0x1	2400 b/s
0x2	4800 b/s
0x3	9600 b/s
0x4	19200 b/s
0x5	38400 b/s
0x6	57600 b/s
0x7	115200 b/s
0x8	230400 b/s
The baud rate limit is 7 Mb/s.	

Default

0x03 (9600 b/s)

NB (Parity)

This command applies to the XBee 868LP RF Module.

Set or read the serial parity settings for UART communications.

Parameter range

0x00 - 0x02

Parameter	Description
0x00	No parity
0x01	Even parity
0x02	Odd parity

Parameter	Description
0	No parity
1	Even parity
2	Odd parity

Default

0x00

SB (Stop Bits)

This command applies to the XBee 868LP RF Module.

Sets or displays the number of stop bits for UART communications.

Parameter range

0 - 1

Parameter	Configuration
0	One stop bit
1	Two stop bits

Default

0

RO (Packetization Timeout)

This command applies to the XBee 868LP RF Module.

Set or read the number of UART character times of inter-character silence required before transmission begins when operating in Transparent mode.

Set **RO** to **0** to transmit characters as they arrive instead of buffering them into one RF packet.

Parameter range

0 - 0xFF (x character times)

Default

3

FT (Flow Control Threshold)

This command applies to the XBee 868LP RF Module.

Set or display the flow control threshold.

The device de-asserts $\overline{\text{CTS}}$ and/or send XOFF when **FT** bytes are in the UART receive buffer. It re-asserts $\overline{\text{CTS}}$ when less than **FT**-16 bytes are in the UART receive buffer.

Parameter range

0x11 - 0x16F bytes

Default

0x13F

AP (API Mode)

This command applies to the XBee 868LP RF Module.

Sets or reads the UART API mode.

Parameter range

0 - 2

The following settings are allowed:

Parameter	Description
0	Transparent mode, API mode is off. All UART input and output is raw data and the device uses the **RO** parameter to delineate packets.
1	API Mode Without Escapes. The device packetizes all UART input and output data in API format, without escape sequences.
2	API Mode With Escapes. The device is in API mode and inserts escaped sequences to allow for control characters. The device passes XON, XOFF, Escape, and the 0x7E delimiter as data.

Default

0

AO (API Options)

This command applies to the XBee 868LP RF Module.

The API data frame output format for RF packets received. This parameter applies to both the UART and SPI interfaces.

Parameter range

0, 1

Parameter	Description
0	API Rx Indicator - 0x90, this is for standard data frames.
1	API Explicit Rx Indicator - 0x91, this is for Explicit Addressing data frames.

Default

0

I/O settings commands

The following AT commands are I/O settings commands.

CB (Commissioning Pushbutton)

This command applies to the XBee 868LP RF Module.

Use **CB** to simulate commissioning pushbutton presses in software.

Set the parameter value to the number of button presses that you want to simulate. For example, send **CB1** to perform the action of pressing the Commissioning Pushbutton once.

See The Commissioning Pushbutton.

See Commissioning pushbutton.

Parameter range

0 - 4

Default

N/A

D0 (AD0/DIO0 Configuration)

This command applies to the XBee 868LP RF Module.

Sets or displays the DIO0/AD0 configuration (pin 33).

Parameter range

0 - 5

Parameter	Description
0	Disabled
1	Commissioning Pushbutton
2	ADC
3	Digital input
4	Digital output, low
5	Digital output, high

Default

1

D1 (DIO1/AD1)

This command applies to the XBee 868LP RF Module.

Sets or displays the DIO1/AD1 configuration (pin 32).

Parameter range

0, 2 - 5

Parameter	Description
0	Disabled
1	Commissioning button
2	ADC
3	Digital input
4	Digital output, low
5	Digital output, high

Default

0

D2 (DIO2/AD2)

This command applies to the XBee 868LP RF Module.

Sets or displays the DIO2/AD2 configuration (pin 31).

Parameter range

0, 2 - 5

Parameter	Description
0	Disabled
2	ADC
3	Digital input
4	Digital output, low
5	Digital output, high

Default

0

D3 (DIO3/AD3)

This command applies to the XBee 868LP RF Module.

Sets or displays the DIO3/AD3 configuration (pin 30).

Parameter range

0, 2 - 5

Parameter	Description
0	Disabled
2	ADC
3	Digital input
4	Digital output, low
5	Digital output, high

Default

0

D4 (DIO4/AD4)

This command applies to the XBee 868LP RF Module.

Sets or displays the DIO4 configuration (pin 24).

Parameter range

0, 3 - 5

Parameter	Description
0	Disabled
3	Digital input
4	Digital output, low
5	Digital output, high

Default

0

D5 (DIO5/ASSOCIATED_INDICATOR)

This command applies to the XBee 868LP RF Module.

Sets or displays the DIO5/AD5/ASSOCIATED_INDICATOR configuration (pin 28).

Parameter range

0, 1, 3 - 5

Parameter	Description
0	Disabled
1	Associate LED indicator - blinks when associated
3	Digital input
4	Digital output, default low
5	Digital output, default high

Default

1

D6 (DIO6/RTS)

This command applies to the XBee 868LP RF Module.

Sets or displays the DIO6/$\overline{\text{RTS}}$ configuration (pin 29).

Parameter range

0, 1, 3 - 5

Parameter	Description
0	Disabled
1	$\overline{\text{RTS}}$ flow control
3	Digital input

Parameter	Description
4	Digital output, low
5	Digital output, high

Default

0

D7 (DIO7/CTS)

This command applies to the XBee 868LP RF Module.

Sets or displays the DIO7/$\overline{\text{CTS}}$ configuration (pin 25).

Parameter range

0, 1, 3 - 7

Parameter	Description
0	Disabled
1	$\overline{\text{CTS}}$ flow control
3	Digital input
4	Digital output, low
5	Digital output, high
6	RS-485 Tx enable, low Tx (0 V on transmit, high when idle)
7	RS-485 Tx enable high, high Tx (high on transmit, 0 V when idle)

Default

0x1

D8 (DIO8/SLEEP_REQUEST)

This command applies to the XBee 868LP RF Module.

Sets or displays the DIO8/SLEEP_REQUEST configuration (pin 10).

Parameter range

0, 1, 3 - 5

Parameter	Description
0	Disabled
1	Sleep request
2	N/A

Parameter	Description
3	Digital input
4	Digital output, low
5	Digital output, high

Default

1

D9 (DIO9/ON_SLEEP)

This command applies to the XBee 868LP RF Module.

Sets or displays the DIO9/ON_SLEEP configuration (pin 26).

Parameter range

0, 1, 3 - 5

Parameter	Description
0	Disabled
1	ON/SLEEP output
3	Digital input
4	Digital output, low
5	Digital output, high

Default

1

P0 (DIO10/RSSI/PWM0 Configuration)

This command applies to the XBee 868LP RF Module.

Sets or displays the PWM0/RSSI/DIO10 configuration (pin 7).

Parameter range

0 - 5

Parameter	Description
0	Disabled
1	RSSI PWM0 output
2	PWM0 output
3	Digital input

Parameter	Description
4	Digital output, low
5	Digital output, high

Default

1

P1 (DIO11/PWM1 Configuration)

This command applies to the XBee 868LP RF Module.

Sets or displays the DIO11/PWM1 configuration (pin 8).

Parameter range

0 - 5

Parameter	Description
0	Disabled
1	32.768 kHz clock output
2	PWM1 output
3	Digital input
4	Digital output, low
5	Digital output, high

Default

0

P2 (DIO12 Configuration)

This command applies to the XBee 868LP RF Module.

Sets or displays the DIO12 configuration (pin 5).

Parameter range

0, 3 - 5

Parameter	Description
0	Disabled
3	Digital input
4	Digital output, low
5	Digital output, high

Default

0

P3 (DIO13/DOUT)

This command applies to the XBee 868LP RF Module.

Sets or displays the DIO13/DOUT configuration (pin 3).

Parameter range

0, 1

Parameter	Description
0	Disabled
1	UART DOUT output

Default

1

P4 (DIO14/DIN)

This command applies to the XBee 868LP RF Module.

Sets or displays the DIO14/DIN configuration (pin 4).

Parameter range

0 - 1

Parameter	Description
0	Disabled
1	UART DIN/input

Default

1

P5 (SPI_MISO)

This command applies to the XBee 868LP RF Module.

Sets or displays the DIO15/SPI_MISO configuration (pin 17).

Parameter range

0, 1

0, 1, 4, 5

Parameter	Description
0	Disabled
1	SPI_MISO
4	Digital output low
5	Digital output high

Default

1

P6 (SPI_MOSI Configuration)

This command applies to the XBee 868LP RF Module.

Sets or displays the DIO16/SPI_MOSI configuration (pin 16).

Parameter range

0, 1, 4, 5

Parameter	Description
0	Disabled
1	SPI_MOSI
4	Digital output low
5	Digital output, high

Default

1

P7 (DIO17/SPI_SSEL)

This command applies to the XBee 868LP RF Module.

Sets or displays the DIO17/SPI_$\overline{\text{SSEL}}$ configuration (pin 15).

Parameter range

0, 1, 4, 5

Parameter	Description
0	Disabled
1	SPI_$\overline{\text{SSEL}}$
4	Digital output low
5	Digital output, high

Default

1

P8 (DIO18/SPI_SCLK)

This command applies to the XBee 868LP RF Module.

Sets or displays the DIO18/SPI_SCLK configuration (pin 14).

Parameter range

0, 1, 4, 5

Parameter	Description
0	Disabled
1	SPI_SCLK
4	Digital output low
5	Digital output high

Default

1

P9 (SPI_ATTN)

This command applies to the XBee 868LP RF Module.

Sets or displays the DIO19/SPI_$\overline{\text{ATTN}}$ configuration (pin 12).

Parameter range

0, 1, 4 - 6

Parameter	Description
0	Disabled
1	SPI_$\overline{\text{ATTN}}$
4	Digital output low
5	Digital output high
6	UART data present indicator

Default

1

PD (Pull Up/Down Direction)

This command applies to the XBee 868LP RF Module.

The resistor pull direction bit field (1 = pull-up, 0 = pull-down) for corresponding I/O lines that are set by the **PR** command.

Parameter range

0x0 - 0xFFFFF (bit field)

Default

0x0

PR (Pull-up/Down Resistor Enable)

This command applies to the XBee 868LP RF Module.

The bit field that configures the internal pull-up/down resistor status for the I/O lines.

- If you set a **PR** bit to 1, it enables the pull-up/down resistor
- If you set a **PR** bit to 0, it specifies no internal pull-up/down resistor.

PR and **PD** only affect lines that are configured as digital inputs or disabled.

The following table defines the bit-field map for **PR** and **PD** commands.

Bit	I/O line	Device pin	Range
0	DIO4/AD4	pin 24	40 kΩ
1	DIO3/AD3	pin 30	40 kΩ
2	DIO2/AD2	pin 31	40 kΩ
3	DIO1/AD1	pin 32	40 kΩ
4	DIO0/AD0	pin 33	40 kΩ
5	DIO6/$\overline{\text{RTS}}$	pin 29	40 kΩ
6	DIO8/SLEEP_RQ/$\overline{\text{DTR}}$	pin 10	40 kΩ
7	DIN/$\overline{\text{CONFIG}}$	pin 4	40 kΩ
8	DIO5/ASSOCIATE	pin 28	40 kΩ
9	DIO9/On/$\overline{\text{SLEEP}}$	pin 26	40 kΩ
10	DIO12	pin 5	40 kΩ
11	DIO10/PWM0/RSSI	pin 7	40 kΩ
12	DIO11/PWM1	pin 8	40 kΩ
13	DIO7/$\overline{\text{CTS}}$	pin 25	40 kΩ
14	DOUT	pin 3	40 kΩ
15	DIO15/SPI_MISO	pin 17	40 kΩ
16	DIO16/SPI_MOSI	pin 16	40 kΩ
17	DIO17/SPI_SSEL	pin 15	40 kΩ
18	DIO18/SPI_SCLK	pin 14	40 kΩ
19	DIO19/SPI_ATTN	pin 12	40 kΩ

Parameter range

0 - 0xFFFFF (bit field)

Default

0xFFFFF

M0 (PWM0 Duty Cycle)

This command applies to the XBee 868LP RF Module.

The duty cycle of the PWM0 line (pin 7).

Use the **P0** command to configure the line as a PWM output.

Parameter range

0 - 0x3FF

Default

0

M1 (PWM1 Duty Cycle)

This command applies to the XBee 868LP RF Module.

The duty cycle of the PWM1 line (pin 8).

Use the **P1** command to configure the line as a PWM output.

Parameter range

0 - 0x3FF

Default

0

LT (Associate LED Blink Time)

This command applies to the XBee 868LP RF Module.

Set or read the Associate LED blink time. If you use the **D5** command to enable the Associate LED functionality (DIO5/Associate pin), this value determines the on and off blink times for the LED when the device has joined the network.

If **LT** = **0**, the device uses the default blink rate: 500 ms for a sleep coordinator, 250 ms for all other nodes.

For all other **LT** values, the firmware measures **LT** in 10 ms increments.

Parameter range

0x14 - 0xFF (x 10 ms)

Default

0

RP (RSSI PWM Timer)

This command applies to the XBee 868LP RF Module.

The PWM timer expiration in 0.1 seconds. **RP** sets the duration of pulse width modulation (PWM) signal output on the RSSI pin.

When **RP** = **0xFF**, the output is always on.

Parameter range

0 - 0xFF (x 100 ms)

Default

0x28 (four seconds)

I/O sampling commands

The following AT commands configure I/O sampling parameters.

AV (Analog Voltage Reference)

This command applies to the XBee 868LP RF Module.

The analog voltage reference used for A/D sampling.

Parameter range

0, 1

Parameter	Description
0	1.25 V reference
1	2.5 V reference

Default

1

IC (DIO Change Detection)

This command applies to the XBee 868LP RF Module.

Set or 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** - **P2**) . If you enable a pin as a digital I/O, you can use the **IC** command to force an immediate I/O sample transmission when the DIO state changes. IC is a bitmask that you can use to enable or disable edge detection on individual channels.

Set unused bits to 0.

Bit	I/O line
0	DIO0
1	DIO1
2	DIO2

Bit	I/O line
3	DIO3
4	DIO4
5	DIO5
6	DIO6
7	DIO7
8	DIO8
9	DIO9
10	DIO10
11	DIO11
12	DIO12

Parameter range

0 - 0xFFFF (bit field)

Default

0

IF (Sleep Sample Rate)

This command applies to the XBee 868LP RF Module.

Set or read the number of sleep cycles that must elapse between periodic I/O samples. This allows the firmware to take I/O samples only during some wake cycles. During those cycles, the firmware takes I/O samples at the rate specified by **IR**.

- D0 (AD0/DIO0 Configuration) through D9 (DIO9/ON_SLEEP)
- P0 (DIO10/RSSI/PWM0 Configuration) through P2 (DIO12 Configuration)

Parameter range

0 - 0xFF

Default

1

IR (I/O Sample Rate)

This command applies to the XBee 868LP RF Module.

Set or read the I/O sample rate to enable periodic sampling.

If you set the I/O sample rate to greater than **0**, the device samples all enabled digital I/O and analog inputs at a specified interval. Samples are sent to the address specified by the **DH** and **DL** commands.

To enable periodic sampling, set **IR** to a non-zero value, and enable the analog or digital I/O functionality of at least one device pin. The sample rate is measured in milliseconds.

WARNING! If you set **IR** to 1 or 2, the device will not keep up and many samples will be lost.

Parameter range

0 - 0xFFFF (x 1 ms)

Default

0

TP (Temperature)

This command applies to the XBee 868LP RF Module.

The current module temperature in degrees Celsius in 8-bit two’s compliment format. For example 0x1A = 26 °C, and 0xF6 = -10 °C.

Parameter range

0 - 0xFF [read-only]

Default

N/A

IS (Force Sample)

This command applies to the XBee 868LP RF Module.

Forces a read of all enabled digital and analog input lines.

Parameter range

N/A

Default

N/A

%V (Voltage Supply Monitoring)

This command applies to the XBee 868LP RF Module.

Displays the supply voltage of the device in mV units.

Parameter range

This is a read-only parameter

Default

N/A

Sleep commands

The following AT commands are sleep commands.

SM (Sleep Mode)

This command applies to the XBee 868LP RF Module.

Sets or displays the sleep mode of the device.

Parameter range

0, 1, 4, 5, 7, 8

Parameter	Description
0	Normal.
1	Pin sleep. In this mode, the sleep/wake state of the module is controlled by the SLEEP_REQUEST line.
4	Asynchronous Cyclic Sleep. In this mode, the device periodically sleeps and wakes based on the **SP** and **ST** commands.
5	Asynchronous cyclic sleep with pin wake-up. In this mode, the device is similar to asynchronous cyclic sleep. The device terminates a sleep period when it detects a falling edge of the SLEEP_REQUEST line.
7	Sleep Support
8	Synchronized Cyclic Sleep

Default

0

SO (Sleep Options)

This command applies to the XBee 868LP RF Module.

Set or read the sleep options bit field of a device. This command is a bitmask.

You cannot set bit 0 and bit 1 at the same time.

Parameter range

0 - 0x13E [bit field]

For synchronous sleep devices, the following sleep bit field options are defined:

Bit	Option
0	Preferred sleep coordinator; setting this bit causes a sleep compatible device to always act as sleep coordinator
1	Non-sleep coordinator; setting this bit causes a device to never act as a sleep coordinator
2	Enable API sleep status messages
3	Disable early wake-up for missed syncs
4	Enable node type equality (disables seniority based on device type)
5	Disable lone coordinator sync repeat

For asynchronous sleep devices, the following sleep bit field options are defined:

Bit	Option
8	Always wake for **ST** time

Default

0x2 (non-sleep coordinator)

SN (Number of Cylcles Between ON_SLEEP)

This command applies to the XBee 868LP RF Module.

Set or read the number of sleep periods value. This command controls the number of sleep periods that must elapse between assertions of the ON_SLEEP line during the wake time of Asynchronous Cyclic Sleep.

During cycles when ON_SLEEP is de-asserted, the device wakes up and checks for any serial or RF data. If it receives any such data, then it asserts the ON_SLEEP line and the device wakes up fully. Otherwise, the device returns to sleep after checking.

This command does not work with synchronous sleep devices.

Parameter range

1 - 0xFFFF

Default

1

SP (Sleep Period)

This command applies to the XBee 868LP RF Module.

Sets or displays the device's sleep time. This command defines the amount of time the device sleeps per cycle.

For a node operating as an Indirect Messaging Coordinator, this command defines the amount of time that it will hold an indirect message for an end device. The coordinator will hold the message for (2.5 * **SP**).

Parameter range

0x1 - 0x15F900 (x 10 ms)

Default

0x190 (4 seconds)

ST (Wake Time)

This command applies to the XBee 868LP RF Module.

Sets or displays the wake time of the device.

For devices in asynchronous sleep, **ST** defines the amount of time that a device stays awake after it receives RF or serial data.

For devices in synchronous sleep, **ST** defines the amount of time that a device stays awake when operating in cyclic sleep mode. The command adjusts the value upwards automatically if it is too small to function properly based on other settings.

Parameter range

0x1 - 0x36EE80 (x 1 ms) (one hour)

Default

0x1F40 (8 seconds)

WH (Wake Host)

This command applies to the XBee 868LP RF Module.

Sets or displays the wake host timer value.

If you set **WH** to a non-zero value, this timer specifies a time in milliseconds that the device delays after waking from sleep before sending data out the UART or transmitting an I/O sample. If the device receives serial characters, the **WH** timer stops immediately.

When in synchronous sleep, the device shortens its sleep period by the **WH** value to ensure it is prepared to communicate when the network wakes up. When in this sleep mode, the device always stays awake for the **WH** time plus the amount of time it takes to transmit a one-hop unicast to another node.

Parameter range

0 - 0xFFFF (x 1 ms)

Default

0

Diagnostic - sleep status/timing commands

The following AT commands are Diagnostic sleep status/timing commands.

SS (Sleep Status)

This command applies to the XBee 868LP RF Module.

Queries a number of Boolean values that describe the device's status.

Bit	Description
0	This bit is true when the network is in its wake state.
1	This bit is true if the node currently acts as a network sleep coordinator.
2	This bit is true if the node ever receives a valid sync message after it powers on.
3	This bit is true if the node receives a sync message in the current wake cycle.
4	This bit is true if you alter the sleep settings on the device so that the node nominates itself and sends a sync message with the new settings at the beginning of the next wake cycle.

Bit	Description
5	This bit is true if you request that the node nominate itself as the sleep coordinator using the Commissioning Pushbutton or the **CB**2 command.
6	This bit is true if the node is currently in deployment mode.
All other bits	Reserved. Ignore all non-documented bits.

Parameter range

[read-only]

Default

0x40

OS (Operating Sleep Time)

This command applies to the XBee 868LP RF Module.

Reads the current network sleep time that the device is synchronized to, in units of 10 milliseconds. If the device has not been synchronized, then **OS** returns the value of **SP**.

If the device synchronizes with a sleeping router network, **OS** may differ from **SP**.

Parameter range

[read-only]

Default

0x190

OW (Operating Wake Time)

This command applies to the XBee 868LP RF Module.

Reads the current network wake time that a device is synchronized to, in 1 ms units.

If the device has not been synchronized, then **OW** returns the value of **ST**.

If the device synchronizes with a sleeping router network, **OW** may differ from **ST**.

Parameter range

[read-only]

Default

0

MS (Missed Sync Messages)

This command applies to the XBee 868LP RF Module.

Reads the number of sleep or wake cycles since the device received a sync message.

Supported in the 80k firmware only.

Parameter range

[read-only]

Default

0

SQ (Missed Sleep Sync Count)

This command applies to the XBee 868LP RF Module.

Counts the number of sleep cycles in which the device does not receive a sleep sync.

Set the value to 0 to reset this value.

When the value reaches 0xFFFF it does not increment anymore.

Parameter range

0 - 0xFFFF

Default

0

Command mode options

The following commands are Command mode option commands.

CC (Command Sequence Character)

This command applies to the XBee 868LP RF Module.

Sets or displays the character the device uses between guard times of the Command mode sequence. The Command mode sequence causes the device to enter Command mode.

Note We recommend using the a value within the rage of 0x20 - 0x7F as those are ASCII characters.

Parameter range

0 - 0xFF

Default

0x2B (the ASCII plus character: **+**)

CT (Command Mode Timeout)

This command applies to the XBee 868LP RF Module.

Sets or displays the Command mode timeout parameter. If a device does not receive any valid commands within this time period, it returns to Idle mode from Command mode.

Parameter range

2 - 0x1770 (x 100 ms)

Default

0x64 (10 seconds)

CN (Exit Command Mode)

This command applies to the XBee 868LP RF Module.

Immediately exits Command Mode and applies pending changes.

Parameter range

N/A

Default

N/A

GT (Guard Times)

This command applies to the XBee 868LP RF Module.

Set the required period of silence before and after the command sequence characters of the Command mode sequence (**GT** + **CC** + **GT**). The period of silence prevents inadvertently entering Command mode.

Parameter range

0x2 - 0x95C (x 1 ms)

Default

0x3E8 (one second)

Firmware commands

The following AT commands are firmware commands.

VL (Version Long)

This command applies to the XBee 868LP RF Module.

Shows detailed version information including the application build date and time.

Parameter range

[read-only]

Default

N/A

VR (Firmware Version)

This command applies to the XBee 868LP RF Module.

Reads the firmware version on a device.

Parameter range

0 - 0xFFFFFFFF [read-only]

Default

Set in firmware

HV (Hardware Version)

This command applies to the XBee 868LP RF Module.

Display the hardware version number of the device.

Parameter range

0 - 0xFFFF [read-only]

Default

Set in firmware

HS (Hardware Series)

This command applies to the XBee 868LP RF Module.

Read the device's hardware series number.

For example, if the device is version S8B, this returns 0x801.

Parameter range

0 - 0xFFFF [read-only]

Default

Set in the firmware

DD (Device Type Identifier)

This command applies to the XBee 868LP RF Module.

Stores the Digi device type identifier value. Use this value to differentiate between multiple XBee devices.

Parameter range

0 - 0xFFFFFFFF

Default

0xC0000

NP (Maximum Packet Payload Bytes)

This command applies to the XBee 868LP RF Module.

Reads the maximum number of RF payload bytes that you can send in a transmission.

Parameter range

0 - 0xFFFF (bytes) [read-only]

Default

0x100

CK (Configuration CRC)

This command applies to the XBee 868LP RF Module.

Displays the cyclic redundancy check (CRC) of the current AT command configuration settings.

This command allows you to detect an unexpected configuration change on a device.

After a firmware update this command may return a different value.

Parameter range

N/A

Default

N/A

Operate in API mode

API mode overview

As an alternative to Transparent operating mode, you can use API operating mode. API mode provides a structured interface where data is communicated through the serial interface in organized packets and in a determined order. This enables you to establish complex communication between devices without having to define your own protocol. The API specifies how commands, command responses and device status messages are sent and received from the device using the serial interface or the SPI interface.

We may add new frame types to future versions of firmware, so build the ability to filter out additional API frames with unknown frame types into your software interface.

API frame format

The firmware supports two API operating modes: without escaped characters and with escaped characters. Use the AP command to enable either mode. To configure a device to one of these modes, set the following AP parameter values:

- **AP** = 1: API operation.
- **AP** = 2: API operation (with escaped characters—only possible on UART).

The API data frame structure differs depending on what mode you choose.

API operation (AP parameter = 1)

The following table shows the data frame structure when you enable **AP** = 1:

Frame fields	Byte	Description
Start delimiter	1	0x7E
Length	2 - 3	Most Significant Byte, Least Significant Byte
Frame data	4 - n	API-specific structure
Checksum	n + 1	1 byte

The firmware silently discards any data it receives prior to the start delimiter. If the device does not receive the frame correctly or if the checksum fails, the device replies with a device status frame indicating the nature of the failure.

API operation-with escaped characters (AP parameter = 2)

This mode is only available on the UART, not on the SPI serial port. The following table shows the data frame structure when you enable **AP** = 2:

Frame fields	Byte	Description	
Start delimiter	1	0x7E	
Length	2 - 3	Most Significant Byte, Least Significant Byte	Characters escaped if needed
Frame data	4 - n	API-specific structure	
Checksum	n + 1	1 byte	

Escape characters

When you are 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:

Byte	Description
0x7E	Frame Delimiter
0x7D	Escape
0x11	XON
0x13	XOFF

Example: Raw serial data 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 previous 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 specifies the total number of bytes included in the frame's data field. Its two-byte value excludes the start delimiter, the length, and the checksum.

Frame data

This field contains the information that a device receives or transmits. The structure of frame data depends on the purpose of the API frame:

Start delimiter	Length		Frame data								Checksum
			API identifier	Identifier-specific Data							
1	2	3	4	5	6	7	8	9	...	n	n+1
0x7E	MSB	LSB	cmdID	cmdData							Single byte

The cmdID frame (API-identifier) indicates which API messages contains the cmdData frame (Identifier-specific data). The device sends multi-byte values big endian format.

The XBee 868LP RF Module supports the following API frames:

API frame names	API ID
AT Command	0x08
AT Command - Queue Parameter Value	0x09
Transmit Request	0x10
Explicit Addressing Command Frame	0x11
Remote Command Request	0x17
AT Command Response	0x88
Modem Status	0x8A
Transmit Status	0x8B
Receive Packet (AO=0)	0x90
Explicit Rx Indicator (AO=1)	0x91
I/O Data Sample RX Indicator	0x92
Node Identification Indicator (AO=0)	0x95
Remote Command Response	0x97

Calculate and verify checksums

To calculate the checksum of an API frame:

1. Add all bytes of the packet, except the start delimiter 0x7E and the length (the second and third bytes).
2. Keep only the lowest 8 bits from the result.
3. Subtract this quantity from 0xFF.

To verify the checksum of an API frame:

1. Add all bytes including the checksum; do not include the delimiter and length.
2. If the checksum is correct, the last two digits on the far right of the sum equal 0xFF.

Example

Consider the following sample data packet: **7E 00 0A 01 01 50 01 00 48 65 6C 6C 6F B8**+

Byte(s)	Description
7E	Start delimiter
00 0A	Length bytes
01	API identifier
01	API frame ID
50 01	Destination address low

Byte(s)	Description
00	Option byte
48 65 6C 6C 6F	Data packet
B8	Checksum

To calculate the check sum you add all bytes of the packet, excluding the frame delimiter **7E** and the length (the second and third bytes):

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

Add these hex bytes:

01 + 01 + 50 + 01 + 00 + 48 + 65 + 6C + 6C + 6F = 247

Now take the result of 0x247 and keep only the lowest 8 bits which in this example is 0x47 (the two far right digits). Subtract 0x47 from 0xFF and you get 0xB8 (0xFF - 0x47 = 0xB8). 0xB8 is the checksum for this data packet.

If an API data packet is composed with an incorrect checksum, the XBee 868LP RF Module will consider the packet invalid and will ignore the data.

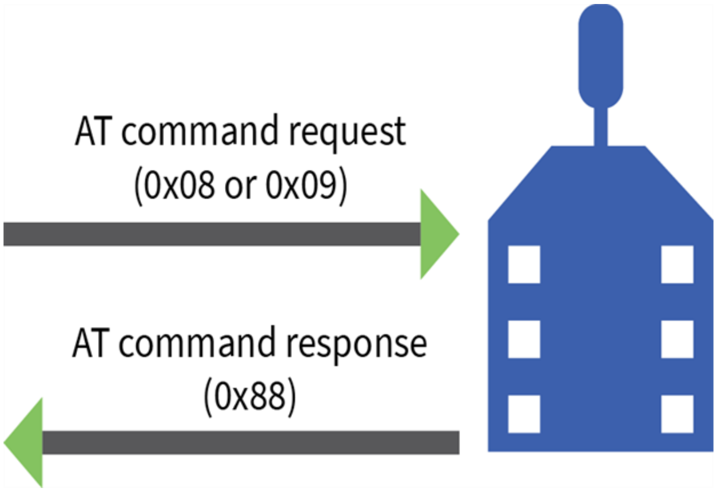
To verify the check sum of an API packet add all bytes including the checksum (do not include the delimiter and length) and if correct, the last two far right digits of the sum will equal FF.

01 + 01 + 50 + 01 + 00 + 48 + 65 + 6C + 6C + 6F + B8 = 2FF

API frame exchanges

AT commands

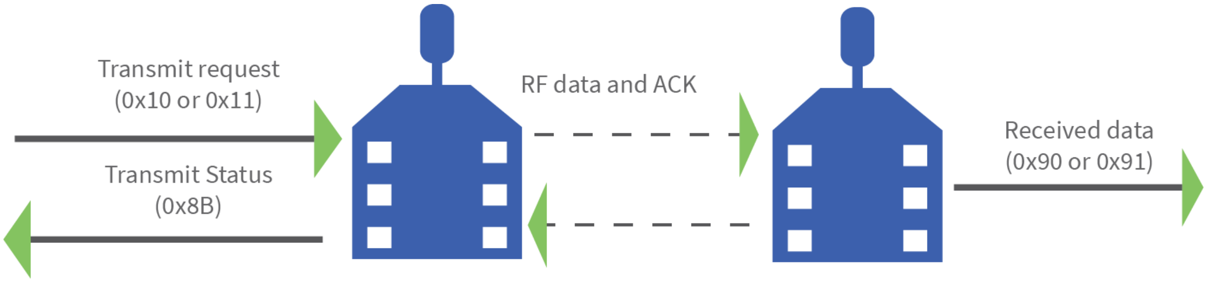
The following image shows the API frame exchange that takes place at the UART when you send a 0x08 AT Command Request or 0x09 AT Command-Queue Request to read or set a device parameter. To disable the 0x88 AT Command Response, set the frame ID to 0 in the request.



Transmit and Receive RF data

The following image shows the API frames exchange that take place at the UART interface 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 TX request. If the packet cannot be delivered to the destination, the transmit status frame indicates the cause of failure.

The received data frame type (0x90 or 0x91) is determined by the **AO** command.

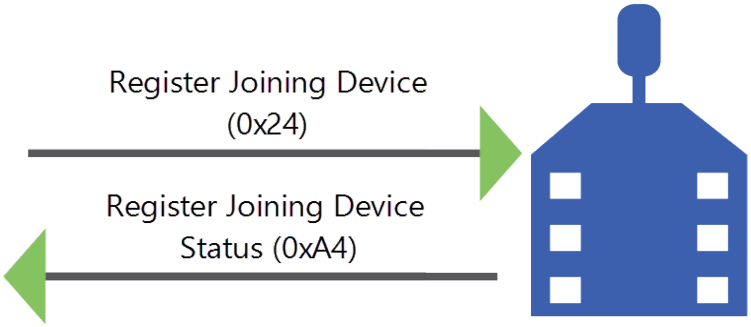


Remote AT commands

The following image shows the API frame exchanges that take place at the serial interface when sending a remote AT command. The device does not send out a remote command response frame through the serial interface if the remote device does not receive the remote command.

Device Registration

The following image shows the API frame exchanges that take place at the serial interface when registering a joining device to a trust center.



Code to support future API frames

If your software application supports the API, you should make provisions that allow for new API frames in future firmware releases. For example, you can include the following section of code on a host microprocessor that handles serial API frames that are sent out the device's DOUT pin:

```
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;
        }
}
```

Frame data

This field contains the information that a device receives or will transmit. The structure of frame data depends on the purpose of the API frame:

Start delimiter	Length		Frame data								Checksum
			Frame type	Data							
1	2	3	4	5	6	7	8	9	...	n	n+1
0x7E	MSB	LSB	API frame type	Data							Single byte

- **Frame type** is the API frame type identifier. It determines the type of API frame and indicates how the Data field organizes the information.
- **Data** contains the data itself. This information and its order depend on the what type of frame that the Frame type field defines.

Multi-byte values are sent big-endian.

Local AT Command Request - 0x08

Response frame: Local AT Command Response - 0x88

Description

This frame type is used to query or set command parameters on the local device. Any parameter that is set with this frame type will apply the change immediately. If you wish to queue multiple parameter changes and apply them later, use the Queue Local AT Command Request - 0x09 instead.

When querying parameter values, this frame behaves identically to Queue Local AT Command Request - 0x09: You can query parameter values by sending this frame with a command but no parameter value field—the two-byte AT command is immediately followed by the frame checksum. When an AT command is queried, a Local AT Command Response - 0x88 frame is populated with the parameter value that is currently set on the device. The Frame ID of the 0x88 response is the same one set by the command in the 0x08 request frame.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Local AT Command Request - **0x08**
4	8-bit	**Frame ID**	Identifies the data frame for the host to correlate with a subsequent response. If set to**0**, the device will not emit a response frame.
5	16-bit	**AT command**	The two ASCII characters that identify the AT Command.
7-n	variable	**Parameter value (optional)**	If present, indicates the requested parameter value to set the given register. If no characters are present, it queries the current parameter value and returns the result in the response.
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Examples

Each example is written without escapes (**AP** = **1**) and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

Set the local command parameter

Set the **NI** string of the radio to "**End Device**".

The corresponding Local AT Command Response - 0x88 with a matching Frame ID will indicate whether the parameter change succeeded.

```
7E 00 0E 08 A1 4E 49 45 6E 64 20 44 65 76 69 63 65 38
```

Frame type	Frame ID	AT command	Parameter value
0x08	0xA1	0x4E49	0x456E6420446576696365
Request	*Matches response*	*"NI"*	*"End Device"*

Query local command parameter

Query the temperature of the module—**TP** command.

The corresponding Local AT Command Response - 0x88 with a matching Frame ID will return the temperature value.

```
7E 00 04 08 17 54 50 3C
```

Frame type	Frame ID	AT command	Parameter value
0x08	0x17	0x5450	(omitted)
Request	*Matches response*	*"TP"*	*Query the parameter*

Queue Local AT Command Request - 0x09

Response frame: Local AT Command Response - 0x88

Description

This frame type is used to query or set queued command parameters on the local device. In contrast to Local AT Command Request - 0x08, this frame queues new parameter values and does not apply them until you either:

- Issue a Local AT Command using the 0x08 frame
- Issue an **AC** command—queued or otherwise

When querying parameter values, this frame behaves identically to Local AT Command Request - 0x08: You can query parameter values by sending this frame with a command but no parameter value field—the two-byte AT command is immediately followed by the frame checksum. When an AT command is queried, a Local AT Command Response - 0x88 frame is populated with the parameter value that is currently set on the device. The Frame ID of the 0x88 response is the same one set by the command in the 0x09 request frame.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Queue Local AT Command Request - **0x09**
4	8-bit	**Frame ID**	Identifies the data frame for the host to correlate with a subsequent response. If set to**0**, the device will not emit a response frame.
5	16-bit	**AT command**	The two ASCII characters that identify the AT Command.
7-n	variable	**Parameter value (optional)**	If present, indicates the requested parameter value to set the given register at a later time. If no characters are present, it queries the current parameter value and returns the result in the response.
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Examples

Each example is written without escapes (**AP** = **1**) and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

Queue setting local command parameter

Set the UART baud rate to 115200, but do not apply changes immediately.

The device will continue to operate at the current baud rate until the change is applied with a subsequent **AC** command.

The corresponding Local AT Command Response - 0x88 with a matching Frame ID will indicate whether the parameter change succeeded.

```
7E 00 05 09 53 42 44 07 16
```

Frame type	Frame ID	AT command	Parameter value
0x09	0x53	0x4244	0x07
Request	*Matches response*	*"BD"*	*7 = 115200 baud*

Query local command parameter

Query the temperature of the module (**TP** command).

The corresponding Local AT Command Response - 0x88 frame with a matching Frame ID will return the temperature value.

```
7E 00 04 09 17 54 50 3B
```

Frame type	Frame ID	AT command	Parameter value
0x09	0x17	0x5450	(omitted)
Request	*Matches response*	*"TP"*	*Query the parameter*

Transmit Request - 0x10

Response frame: Extended Transmit Status - 0x8B

Description

This frame type is used to send payload data as an RF packet to a specific destination. This frame type is typically used for transmitting serial data to one or more remote devices.

The endpoints used for these data transmissions are defined by the **SE** and **EP** commands and the cluster ID defined by the **CI** command—excluding 802.15.4. To define the application-layer addressing fields on a per-packet basis, use the Explicit Addressing Command Request - 0x11 instead.

Query the **NP** command to read the maximum number of payload bytes that can be sent.

64-bit addressing

- For broadcast transmissions, set the 64-bit destination address to **0x000000000000FFFF**
- For unicast transmissions, set the 64-bit address field to the address of the desired destination node

Format

The following table provides the contents of the frame. For details on the frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Transmit Request - **0x10**
4	8-bit	**Frame ID**	Identifies the data frame for the host to correlate with a subsequent response frame. If set to **0**, the device will not emit a response frame.
5	64-bit	**64-bit destination address**	Set to the 64-bit IEEE address of the destination device. Broadcast address is **0x000000000000FFFF**.
13	16-bit	**Reserved**	Unused, but this field is typically set to **0xFFFE**.
15	8-bit	**Broadcast radius**	Sets the maximum number of hops a broadcast transmission can traverse. This parameter is only used for broadcast transmissions. If set to**0**—recommended—the value of **NH**specifies the broadcast radius.
16	8-bit	**Transmit options**	See the Transmit options bit field table below for available options. If set to **0**, the value of **TO** specifies the transmit options.

Offset	Size	Frame Field	Description
17-n	variable	**Payload data**	Data to be sent to the destination device. Up to **NP** bytes per packet.
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Transmit options bit field

The available transmit options vary depending on the protocol being used. Bitfield options can be combined. Set all unused bits to **0**.

Examples

Each example is written without escapes (**AP**=**1**) and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

64-bit unicast

Sending a unicast transmission to a device with the 64-bit address of **0013A20012345678** with the serial data "**TxData**". Transmit options are set to **0**, which means the transmission will send using the options set by the **TO** command.

The corresponding Transmit Status - 0x89 response with a matching Frame ID will indicate whether the transmission succeeded.

```
7E 00 14 10 52 00 13 A2 00 12 34 56 78 FF FE 00 00 54 78 44 61 74 61 91
```

Frame type	Frame ID	64-bit dest	Reserved	Bcast radius	Options	RF data
0x10	0x52	0x0013A200 12345678	0xFFFE	0x00	0x00	0x547844617461
Request	*Matches response*	*Destination*	*Unused*	*N/A*	*Will use **TO***	*"TxData"*

64-bit broadcast

Sending a broadcast transmission of the serial data "**Broadcast**" to neighboring devices and suppressing the corresponding response by setting Frame ID to **0**.

```
7E 00 17 10 00 00 00 00 00 00 00 00 FF FF FF FE 01 00 42 72 6F 61 64 63 61 73
74 60
```

Frame type	Frame ID	64-bit dest	Reserved	Bcast radius	Tx Options	RF data
0x10	0x00	0x00000000 0000FFFF	0xFFFE	0x01	0x00	0x42726F616463617374
Request	*Suppress response*	*Broadcast address*	*Unused*	*Single hop broadcast*	*Will use **TO***	*"Broadcast"*

Explicit Addressing Command Request - 0x11

Response frame: Extended Transmit Status - 0x8B

Description

This frame type is used to send payload data as an RF packet to a specific destination using application-layer addressing fields. The behavior of this frame is similar to Transmit Request - 0x10, but with additional fields available for user-defined endpoints, cluster ID, and profile ID.

This frame type is typically used for OTA updates, and serial data transmissions.

Query NP (Maximum Packet Payload Bytes) to read the maximum number of payload bytes that can be sent.

64-bit addressing

- For broadcast transmissions, set the 64-bit destination address to **0x000000000000FFFF**
- For unicast transmissions, set the 64-bit address field to the address of the desired destination node

Reserved endpoints

For serial data transmissions, the **0xE8** endpoint should be used for both source and destination endpoints.

The active Digi endpoints are:

- **0xE8** - Digi Data endpoint
- **0xE6** - Digi Device Object (DDO) endpoint

Reserved cluster IDs

For serial data transmissions, the **0x0011** cluster ID should be used.

The following cluster IDs can be used on the **0xE8** data endpoint:

- **0x0011**- Transparent data cluster ID
- **0x0012** - Loopback cluster ID:The destination node echoes any transmitted packet back to the source device. Cannot be used on XBee 802.15.4 firmware.

Reserved profile IDs

The Digi profile ID of **0xC105** should be used when sending serial data between XBee devices.

Format

The following table provides the contents of the frame. For details on the frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.

Offset	Size	Frame Field	Description
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Explicit Addressing Command Request - **0x11**
4	8-bit	**Frame ID**	Identifies the data frame for the host to correlate with a subsequent response. If set to**0**, the device will not emit a response frame.
5	64-bit	**64-bit destination address**	Set to the 64-bit IEEE address of the destination device. Broadcast address is **0x000000000000FFFF**. When using 16-bit addressing, set this field to **0xFFFFFFFFFFFFFFFF**.
13	16-bit	**Reserved**	Unused, but this field is typically set to **0xFFFE**.
15	8-bit	**Source Endpoint**	Source endpoint for the transmission. Serial data transmissions should use **0xE8**.
16	8-bit	**Destination Endpoint**	Destination endpoint for the transmission. Serial data transmissions should use **0xE8**.
17	16-bit	**Cluster ID**	The Cluster ID that the host uses in the transmission. Serial data transmissions should use **0x11**.
19	16-bit	**Profile ID**	The Profile ID that the host uses in the transmission. Serial data transmissions between XBee devices should use **0xC105**.
21	8-bit	**Broadcast radius**	Sets the maximum number of hops a broadcast transmission can traverse. This parameter is only used for broadcast transmissions. If set to **0** (recommended), the value of **NH** specifies the broadcast radius.
22	8-bit	**Transmit options**	See the Transmit options bit field table below for available options. If set to **0**, the value of **TO** specifies the transmit options.
23-n	variable	**Command data**	Data to be sent to the destination device. Up to **NP** bytes per packet.
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Transmit options bit field

The available transmit options vary depending on the protocol being used. Bitfield options can be combined. Set all unused bits to **0**.

Examples

Each example is written without escapes (**AP** = **1**) and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

64-bit unicast

Sending a unicast transmission to an XBee device with the 64-bit address of **0013A20012345678** with the serial data "**TxData**". Transmit options are set to **0**, which means the transmission will send using the options set by the **TO** command. This transmission is identical to a Transmit Request - 0x10 using default settings.

The corresponding Extended Transmit Status - 0x8B response with a matching Frame ID will indicate whether the transmission succeeded.

```
7E 00 1A 11 87 00 13 A2 00 12 34 56 78 FF FE E8 E8 00 11 C1 05 00 00 54 78 44
61 74 61 B4
```

Frame type	Frame ID	64-bit dest	Reserved	Source EP	Dest EP	Cluster	Profile	Bcast radius	Tx options	Command data
0x11	0x87	0x0013A200 12345678	0xFFFE	0xE8	0xE8	0x0011	0xC105	0x00	0x00	0x547844617461
Explicit request	*Matches response*	*Destination*	*Unused*	*Digi data*	*Digi data*	*Data*	*Digi profile*	*N/A*	*Use **TO***	*"TxData"*

Loopback Packet

Sending a loopback transmission to an device with the 64-bit address of **0013A20012345678** using Cluster ID **0x0012**. To better understand the raw performance, retries and acknowledgements are disabled.

The corresponding Extended Transmit Status - 0x8B response with a matching Frame ID can be used to verify that the transmission was sent.

The destination will not emit a receive frame, instead it will return the transmission back to the sender. The source device will emit the receive frame—the frame type is determined by the value of **AO**—if the packet looped back successfully.

```
7E 00 1A 11 F8 00 13 A2 00 12 34 56 78 FF FE E8 E8 00 12 C1 05 00 01 54 78 44
61 74 61 41
```

Frame type	Frame ID	64-bit dest	Reserved	Source EP	Dest EP	Cluster	Profile	Bcast radius	Tx options	Command data
0x11	0xF8	0x0013A200 12345678	0xFFFE	0xE8	0xE8	0x0012	0xC105	0x00	0x01	0x547844617461
Explicit request	*Matches response*	*Destination*	*Unused*	*Digi data*	*Digi data*	*Data*	*Digi profile*	*N/A*	*Disable retries*	*"TxData"*

Remote AT Command Request - 0x17

Response frame: Remote AT Command Response- 0x97

Description

This frame type is used to query or set AT command parameters on a remote device.

For parameter changes on the remote device to take effect, you must apply changes, either by setting the **Apply Changes** options bit, or by sending an **AC** command to the remote.

When querying parameter values you can query parameter values by sending this framewith a command but no parameter value field—the two-byte AT command is immediately followed by the frame checksum. When an AT command is queried, a Remote AT Command Response- 0x97 frame is populated with the parameter value that is currently set on the device. The Frame ID of the 0x97 response is the same one set by the command in the 0x17 request frame.

Note Remote AT Command Requests should only be issued as unicast transmissions to avoid potential network disruption. Broadcasts are not acknowledged, so there is no guarantee all devices will receive the request. Responses are returned immediately by all receiving devices, which can cause congestion on a large network.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Remote AT Command Request - **0x17**.
4	8-bit	**Frame ID**	Identifies the data frame for the host to correlate with a subsequent response. If set to **0**, the device will not emit a response frame.
5	64-bit	**64-bit destination address**	Set to the 64-bit IEEE address of the destination device.
13	16-bit	**Reserved**	Unused, but this field is typically set to **0xFFFE**.
15	8-bit	**Remote command options**	Bit field of options that apply to the remote AT command request: ■ **Bit 0**: Disable ACK [**0x01**] ■ **Bit 1**: Apply changes on remote [**0x02**] • If not set, changes will not applied until the device receives an **AC** command or a subsequent command change is received with this bit set

Offset	Size	Frame Field	Description
			■ Bit 2: Reserved (set to 0) ■ Bit 3: Reserved (set to 0) ■ **Bit 4**: Send the remote command securely [**0x10**] **Note** Option values may be combined. Set all unused bits to 0.
16	16-bit	**AT command**	The two ASCII characters that identify the AT Command.
18-n	variable	**Parameter value (optional)**	If present, indicates the requested parameter value to set the given register. If no characters are present, it queries the current parameter value and returns the result in the response.
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Examples

Each example is written without escapes—**AP** = **1**—and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

Set remote command parameter

Set the **NI** string of a device with the 64-bit address of **0013A20012345678** to "**Remote**" and apply the change immediately.

The corresponding Remote AT Command Response- 0x97 with a matching Frame ID will indicate success.

```
7E 00 15 17 27 00 13 A2 00 12 34 56 78 FF FE 02 4E 49 52 65 6D 6F 74 65 F6
```

Frame type	Frame ID	64-bit dest	Reserved	Command options	AT command	Parameter value
0x17	0x27	0x0013A200 12345678	0xFFFE	0x02	0x4E49	0x52656D6F7465
Request	*Matches response*		*Unused*	*Apply Change*	*"NI"*	*"Remote"*

Queue remote command parameter change

Change the PAN ID of a remote device so it can migrate to a new PAN, since this change would cause network disruption, the change is queued so that it can be made active later with a subsequent **AC** command or written to flash with a queued **WR** command so the change will be active after a power cycle.

The corresponding Remote AT Command Response- 0x97 with a matching Frame ID will indicate success.

```
7E 00 11 17 68 00 13 A2 00 12 34 56 78 FF FE 00 49 44 04 51 D8
```

Frame type	Frame ID	64-bit dest	Reserved	Command options	AT command	Parameter value
0x17	0x68	0x0013A200 12345678	0xFFFE	0x00	0x4944	0x0451
Request	*Matches response*		*Unused*	*Queue Change*	*"ID"*	

Query remote command parameter

Query the temperature of a remote device—**TP** command.

The corresponding Remote AT Command Response- 0x97 with a matching Frame ID will return the temperature value.

```
7E 00 0F 17 FA 00 13 A2 00 12 34 56 78 FF FE 00 54 50 84
```

Frame type	Frame ID	64-bit dest	Reserved	Command options	AT command	Parameter value
0x17	0xFA	0x0013A200 12345678	0xFFFE	0x00	0x5450	(omitted)
Request	*Matches response*		*Unused*	*N/A*	*"TP"*	*Query the parameter*

Local AT Command Response - 0x88

Request frames:

- Local AT Command Request - 0x08
- Queue Local AT Command Request - 0x09

Description

This frame type is emitted in response to a local AT Command request. Some commands send back multiple response frames; for example, ND (Network Discover). Refer to individual AT command descriptions for details on API response behavior.

This frame is only emitted if the Frame ID in the request is non-zero.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Local AT Command Response - **0x88**
4	8-bit	**Frame ID**	Identifies the data frame for the host to correlate with a prior request.
5	16-bit	**AT command**	The two ASCII characters that identify the AT Command.
7	8-bit	**Command status**	Status code for the host's request: **0** = OK **1** = ERROR **2** = Invalid command **3** = Invalid parameter
8-n	variable	**Command data (optional)**	If the host requested a command parameter change, this field will be omitted. If the host queried a command by omitting the parameter value in the request, this field will return the value currently set on the device.
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Examples

Each example is written without escapes (**AP** = **1**) and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

Set local command parameter

Host set the NI string of the local device to "**End Device**" using a 0x08 request frame.

The corresponding Local AT Command Response - 0x88 with a matching Frame ID is emitted as a response:

```
7E 00 05 88 01 4E 49 00 DF
```

Frame type	Frame ID	AT command	Command Status	Command data
0x88	0xA1	0x4E49	0x00	(omitted)
Response	*Matches request*	*"NI"*	*Success*	*Parameter changes return no data*

Query local command parameter

Host queries the temperature of the local device—**TP** command—using a 0x08 request frame.

The corresponding Local AT Command Response - 0x88 with a matching Frame ID is emitted with the temperature value as a response:

```
7E 00 07 88 01 54 50 00 FF FE D5
```

Frame type	Frame ID	AT command	Command Status	Command data
0x88	0x17	0x5450	0x00	0xFFFE
Response	*Matches request*	*"TP"*	*Success*	*-2 ℃*

Modem Status - 0x8A

Description

This frame type is emitted in response to specific conditions. The status field of this frame indicates the device behavior.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Modem Status - **0x8A**
4	8-bit	**Modem status**	Complete list of modem statuses: **0x00** = Hardware reset or power up **0x01** = Watchdog timer reset **0x02** = Joined network **0x03** = Left network **0x06** = Coordinator started **0x07** = Network security key was updated **0x0B** = Network woke up **0x0C** = Network went to sleep **0x0D** = Voltage supply limit exceeded **0x0E** = Digi Remote Manager connected **0x0F** = Digi Remote Manager disconnected **0x11** = Modem configuration changed while join in progress **0x12** = Access fault **0x13** = Fatal error **0x3B** = Secure session successfully established **0x3C** = Secure session ended **0x3D** = Secure session authentication failed **0x3E** = Coordinator detected a PAN ID conflict but took no action **0x3F** = Coordinator changed PAN ID due to a conflict **0x32** = BLE Connect **0x33** = BLE Disconnect **0x34** = Bandmask configuration failed **0x35** = Cellular component update started **0x36** = Cellular component update failed **0x37** = Cellular component update completed **0x38** = XBee firmware update started **0x39** = XBee firmware update failed **0x3A** = XBee firmware update applying **0x40** = Router PAN ID was changed by coordinator due to a conflict

Offset	Size	Frame Field	Description
			0x42 = Network Watchdog timeout expired **0x80** through **0xFF** = Stack error Refer to the tables below for a filtered list of status codes that are appropriate for specific devices.
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Modem status codes

Statuses for specific modem types are listed here.

Examples

Each example is written without escapes (**AP** = **1**) and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

Boot status

When a device powers up, it returns the following API frame:

```
7E 00 02 8A 00 75
```

Frame type	Modem Status
0x8A	0x00
Status	*Hardware Reset*

Extended Transmit Status - 0x8B

Request frames:

- Transmit Request - 0x10
- Explicit Addressing Command Request - 0x11

Description

This frame type is emitted when a network transmission request completes. The status field of this frame indicates whether the request succeeded or failed and the reason. This frame type provides additional networking details about the transmission.

This frame is only emitted if the Frame ID in the request is non-zero.

Note Broadcast transmissions are not acknowledged and always return a status of **0x00**, even if the delivery failed.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Transmit Status - **0x8B**
4	8-bit	**Frame ID**	Identifies the data frame for the host to correlate with a prior request.
5	16-bit	**Reserved**	Unused, but this field is typically set to **0xFFFE**.
7	8-bit	**Transmit retry count**	The number of application transmission retries that occur.
8	8-bit	**Delivery status**	Complete list of delivery statuses: **0x00** = Success **0x01** = MAC ACK failure **0x02** = CCA/LBT failure **0x03** = Indirect message unrequested / no spectrum available **0x21** = Network ACK failure **0x25** = Route not found **0x31** = Internal resource error **0x32** = Resource error lack of free buffers, timers, etc. **0x74** = Data payload too large **0x75** = Indirect message unrequested
9	8-bit	**Discovery status**	Complete list of delivery statuses: **0x00** = No discovery overhead

Offset	Size	Frame Field	Description
			0x02 = Route discovery
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Route Information - 0x8D

Request frames:

- Transmit Request - 0x10
- Explicit Addressing Command Request - 0x11

Description

This frame type contains the DigiMesh routing information for a remote device on the network. This route information can be used to diagnose marginal links between devices across multiple hops.

This frame type is emitted in response to a DigiMesh unicast transmission request which has Trace Routing or NACK enabled. See Trace route option and NACK messages for more information.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Route Information - **0x8D**
4	8-bit	**Source event**	Event that caused the route information to be generated: **0x11** = NACK **0x12** = Trace route
5	8-bit	**Data length**	The number of bytes that follow, excluding the checksum. If the length increases, new items have been added to the end of the list for future revisions.
6	32-bit	**Timestamp**	System timer value on the node generating the Route Information Packet.The timestamp is in microseconds. Only use this value for relative time measurements because the time stamp count restarts approximately every hour.
10	8-bit	**ACK timeout count**	The number of MAC ACK timeouts that occur.
11	8-bit	**TX blocked count**	The number of times the transmission was blocked due to reception in progress.
12	8-bit	**Reserved**	Not used.
14	64-bit	**Destination address**	The 64-bit IEEE address of the final destination node of this network-level transmission.
21	64-bit	**Source address**	The 64-bit IEEE address of the source node of this network-level transmission.

Offset	Size	Frame Field	Description
29	64-bit	**Responder address**	The 64-bit IEEE address of the node that generates this Route Information packet after it sends (or attempts to send) the data packet to the next hop (the Receiver node).
37	64-bit	**Receiver address**	The 64-bit IEEE address of the node that the device sends (or attempts to send) the data packet.
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Examples

Each example is written without escapes (**AP** = **1**) and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

Routing information

The following example represents a possible Route Information Packet. A device emits this frame when it performs a trace route enabled transmission from one device—serial number 0x0013A200 4052AAAA—to another—serial number 0x0013A200 4052DDDD—across a DigiMesh network.

This particular frame indicates that the network successfully forwards the transmission from one device—serial number 0x0013A200 4052BBBB—to another device—serial number 0x0013A200 4052CCCC.

```
7E 00 2A 8D 12 27 6B EB CA 93 00 00 00 00 13 A2 00 40 52 DD DD 00 13 A2 00 40
52 AA AA 00 13 A2 00 40 52 BB BB 00 13 A2 00 40 52 CC CC 4E
```

Frame type	Source event	Data length	Timestamp	ACK timeout	TX Blocked	Reserved	Dest address	Source address	Responder address	Receiver address
0x8D	0x12	0x27	0x6BEBCA93	0x00	0x00	0x00	0x0013A200 4052DDDD	0x0013A200 4052AAAA	0x0013A200 4052BBBB	0x0013A200 4052CCCC
Route	*Trace Route*		*~30 minutes*	*No retries this hop*	*No error*	*N/A*	*Destination*	*Source*	*Node that sent this information*	*Next hop*

Aggregate Addressing Update - 0x8E

Description

This frame type is emitted on devices that update it addressing information in response to a network aggregator issuing an addressing update. A network aggregator is defined by a device on the network who has had the AG (Aggregator Support) command issued. A device on the network who's current **DH** and **DL** matches the address provided in the **AG** command request will update **DH** and **DL** and emit this frame.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Aggregate Addressing Update - **0x8E**
4	8-bit	**Reserved**	Reserved for future functionality. This field returns 0.
5	64-bit	**New address**	Address to which **DH** and **DL** are being set.
13	64-bit	**Old address**	Address to which **DH** and **DL** were previously set.
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Examples

Each example is written without escapes (**AP** = **1**) and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

Aggregate address update

In the following example, a device with destination address (**DH**/**DL**) of 0x0013A200 4052AAAA updates its destination address to 0x0013A200 4052BBBB.

```
7E 00 12 8E 00 00 13 A2 00 40 52 BB BB 00 13 A2 00 40 52 AA AA 19
```

Frame type	Reserved	New address	Old address
0x8E	0x00	0x0013A200	0x0013A200

Frame type	Reserved	New address	Old address
		4052BBBB	4052AAAA
Update	*N/A*	*What **DH**/**DL** is now set to*	*What **DH**/**DL** was set to*

Receive Packet - 0x90

Request frames:

- Transmit Request - 0x10
- Explicit Addressing Command Request - 0x11

Description

This frame type is emitted when a device configured with standard API output—AO (API Options) = **0**—receives an RF data packet.

Typically this frame is emitted as a result of a device on the network sending serial data using the Transmit Request - 0x10 or Explicit Addressing Command Request - 0x11 addressed either as a broadcast or unicast transmission.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Receive Packet - **0x90**
4	64-bit	**64-bit source address**	The sender's 64-bit address.
12	16-bit	**Reserved**	Unused, but this field is typically set to **0xFFFE**.
14	8-bit	**Receive options**	Bit field of options that apply to the received message: ■ **Bit 0**: Packet was Acknowledged [**0x01**] ■ **Bit 1**: Packet was sent as a broadcast [**0x02**] ■ **Bit 2**: Reserved ■ **Bit 3**: Reserved ■ **Bit 4**: Reserved ■ **Bit 5**: Reserved ■ **Bit 6**: Reserved ■ **Bit 6, 7**: DigiMesh delivery method • b’00 = <invalid option> • b’01 = Point-multipoint [**0x40**] • b’10 = Directed Broadcast [**0x80**] • b’11 = DigiMesh [**0xC0**]

Offset	Size	Frame Field	Description
			Note Option values may be combined.
15-n	variable	**Received data**	The RF payload data that the device receives.
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Examples

Each example is written without escapes (**AP** = **1**) and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

64-bit unicast

A device with the 64-bit address of **0013A20041AEB54E** sent a unicast transmission to a specific device with the payload of "**TxData**". The following frame is emitted if the destination is configured with **AO** = **0**.

```
7E 00 12 90 00 13 A2 00 41 AE B5 4E FF FE C1 54 78 44 61 74 61 C4
```

Frame type	64-bit source	Reserved	Rx options	Received data
0x90	0x0013A200 41AEB54E	0x5614	0xC1	0x547844617461
Output		*Unused*	*ACK was sent in DigiMesh mode*	*"TxData"*

Explicit Receive Indicator - 0x91

Request frames:

- Transmit Request - 0x10
- Explicit Addressing Command Request - 0x11

Description

This frame type is emitted when a device configured with explicit API output—AO (API Options) bit1 set—receives a packet.

Typically this frame is emitted as a result of a device on the network sending serial data using the Transmit Request - 0x10 or Explicit Addressing Command Request - 0x11 addressed either as a broadcast or unicast transmission.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Explicit Receive Indicator - **0x91**
4	64-bit	**64-bit source address**	The sender's 64-bit address.
12	16-bit	**Reserved**	Unused, but this field is typically set to **0xFFFE**.
14	8-bit	**Source endpoint**	Endpoint of the source that initiated transmission.
15	8-bit	**Destination endpoint**	Endpoint of the destination that the message is addressed to.
16	16-bit	**Cluster ID**	The Cluster ID that the frame is addressed to.
18	16-bit	**Profile ID**	The Profile ID that the fame is addressed to.
20	8-bit	**Receive options**	Bit field of options that apply to the received message for packets sent using Digi endpoints (0xDC-0xEE): ▪ **Bit 0**: Packet was Acknowledged [**0x01**] ▪ **Bit 1**: Packet was sent as a broadcast [**0x02**] ▪ **Bit 2**: Reserved ▪ **Bit 3**: Reserved ▪ **Bit 4**: Reserved

Offset	Size	Frame Field	Description
			■ **Bit 5**: Reserved ■ **Bit 6**: Reserved ■ **Bit 6, 7**: DigiMesh delivery method • b’00 = <invalid option> • b’01 = Point-multipoint [**0x40**] • b’10 = Directed Broadcast [**0x80**] • b’11 = DigiMesh [**0xC0**] **Note** Option values may be combined.
21-n	variable	**Received data**	The RF payload data that the device receives.
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Examples

Each example is written without escapes (**AP** = **1**) and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

64-bit unicast

A device with the 64-bit address of **0013A20087654321** sent a unicast transmission to a specific device with the payload of "**TxData**". The following frame is emitted if the destination is configured with **AO** > **1**.

```
7E 00 18 91 00 13 A2 00 41 AE B5 4E FF FE E8 E8 00 11 C1 05 C1 54 78 44 61 74
61 1C
```

Frame type	64-bit source	Reserved	Source EP	Dest EP	Cluster	Profile	Rx options	Received data
0x91	0x0013A200 41AEB54E	0x87BD	0xE8	0xE8	0x0011	0xC105	0xC1	0x54784461746 1
Explicit output		*Unused*	*Digi data*	*Digi data*	*Data*	*Digi profile*	*ACK was sent in DigiMesh network*	*"TxData"*

I/O Sample Indicator - 0x92

Description

This frame type is emitted when a device configured with standard API output—AO (API Options) = **0**—receives an I/O sample frame from a remote device. Only devices running in API mode will send I/O samples out the serial port.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	I/O Sample Indicator - **0x92**
4	64-bit	**64-bit source address**	The sender's 64-bit IEEE address.
12	16-bit	**Reserved**	Unused, but typically **0XFFFE**.
14	8-bit	**Receive options**	Bit field of options that apply to the received message: ■ **Bit 0**: Packet was Acknowledged [**0x01**] ■ **Bit 1**: Packet was sent as a broadcast [**0x02**] Note Option values may be combined.
15	8-bit	**Number of samples**	The number of sample sets included in the payload. This field typically reports 1 sample.
16	16-bit	**Digital sample mask**	Bit field that indicates which I/O lines on the remote are configured as digital inputs or outputs, if any: **bit 0**: DIO0 **bit 1**: DIO1 **bit 2**: DIO2 **bit 3**: DIO3 **bit 4**: DIO4 **bit 5**: DIO5 **bit 6**: DIO6 **bit 7**: DIO7 **bit 8**: DIO8 **bit 9**: DIO9 **bit 10**: DIO10 **bit 11**: DIO11 **bit 12**: DIO12 **bit 13**: DIO13 **bit 14**: DIO14 bit 15: N/A For example, a digital channel mask of **0x002F** means DIO **0**, **1**, **2**, **3**, and **5** are enabled as digital I/O.
18	8-bit	**Analog sample**	Bit field that indicates which I/O lines on the remote are configured as analog input, if any:

Offset	Size	Frame Field	Description
		mask	**bit 0**: AD0 **bit 1**: AD1 **bit 2**: AD2 **bit 3**: AD3 **bit 7**: Supply Voltage (enabled with **V+** command)
19	16-bit	**Digital samples (if included)**	If the sample set includes any digital I/O lines (**Digital channel mask** > **0**), this field contain samples for all enabled digital I/O lines. If no digital lines are configured as inputs or outputs, this field will be omitted. DIO lines that do not have sampling enabled return 0. Bits in this field are arranged the same as they are in the Digital channel mask field.
22	16-bit variable	**Analog samples (if included)**	If the sample set includes any analog I/O lines (Analog channel mask > 0), each enabled analog input returns a 16-bit value indicating the ADC measurement of that input. Analog samples are ordered sequentially from AD0 to AD3.
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Examples

Each example is written without escapes (**AP** = **1**) and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

I/O sample

A device with the 64-bit address of **0013A20012345678** is configured to periodically send I/O sample data to a particular device. The device is configured with DIO3, DIO4, and DIO5 configured as digital I/O, and AD1 and AD2 configured as an analog input.

The destination will emit the following frame:

```
7E 00 16 92 00 13 A2 00 12 34 56 78 FF FE C1 01 00 38 06 00 28 02 25 00 F8 E8
```

Frame type	64-bit source	Reserved	Rx options	Num samples	Digital channel mask	Analog channel mask	Digital samples	Analog sample 1	Analog sample 2
0x92	0x0013A200 12345678	0x87AC	0xC1	0x01	0x0038	0x06	0x0028	0x0225	0x00F8
Sample		*Unused*	*ACK was sent in mesh network*	*Single sample (typical)*	*b'00* ***111****000 DIO3, DIO4, and DIO5 enabled*	*b'0****11****0 AD1 and AD2 enabled*	*b'00* ***101****000 DIO3 and DIO5 are HIGH;*	*AD1 data*	*AD2 data*

Frame type	64-bit source	Reserved	Rx options	Num samples	Digital channel mask	Analog channel mask	Digital samples	Analog sample 1	Analog sample 2
							DI04 is LOW		

Node Identification Indicator - 0x95

Description

This frame type is emitted when a node identification broadcast is received. The node identification indicator contains information about the identifying device, such as address, identifier string (**NI**), and other relevant data.

A node identifies itself to the network under these conditions:

- The commissioning button is pressed once.
- A **CB 1** command is issued.
- A synchronous sleep node stays awake for 30 seconds in order to receive a sync message. It also sends out an identifying message.

See ND (Network Discover) for information on the payload formatting.

See NO (Node Discovery Options) for configuration options that modify the output of this frame.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Node Identification Indicator - **0x95**
4	64-bit	**64-bit source address**	The sender's 64-bit address.
12	16-bit	**Reserved**	Unused, but this field is typically set to **0xFFFE**.
14	8-bit	**Options**	Bit field of options that apply to the received message: ■ Bit 0: Reserved ■ **Bit 1**: Packet was sent as a broadcast [**0x02**] ■ **Bit 2**: Reserved ■ Bit 4: Reserved ■ Bit 5: Reserved ■ **Bit 6, 7**: DigiMesh delivery method • b’00 = <invalid option> • b’01 = Point-multipoint [**0x40**] • b’10 = Directed Broadcast [**0x80**] • b’11 = DigiMesh [**0xC0**]

Offset	Size	Frame Field	Description
			Note Option values may be combined.
15	16-bit	**Reserved**	Unused, but this field is typically set to **0xFFFE**.
17	64-bit	**64-bit remote address**	The 64-bit address of the device that sent the Node Identification.
25	variable (2-byte minimum)	**Node identification string**	Node identification string on the remote device set by NI (Node Identifier). The identification string is terminated with a NULL byte (0x00).
27+NI	16-bit	**Reserved**	Unused, but this field is typically set to **0xFFFE**.
29+NI	8-bit	**Network device type**	What type of network device the remote identifies as: 0 = Coordinator 1 = Router 2 = End Device
30+NI	8-bit	**Source event**	The event that caused the node identification broadcast to be sent. 0 = Reserved 1 = Frame sent by node identification pushbutton event—see D0 (AD0/DIO0 Configuration).
31+NI	16-bit	**Digi Profile ID**	The Digi application Profile ID—**0xC105**.
33+NI	16-bit	**Digi Manufacturer ID**	The Digi Manufacturer ID—**0x101E**.
35+NI	32-bit	**Device type identifier (optional)**	The user-defined device type on the remote device set by DD (Device Type Identifier). Only included if the receiving device has the appropriate NO (Node Discovery Options) bit set.
EOF-1	8-bit	**RSSI (optional)**	The RSSI of the last hop that relayed the message. Only included if the receiving device has the appropriate NO (Node Discovery Options) bit set.
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte—between length and checksum.

Examples

Each example is written without escapes (**AP** = **1**) and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

Identify remote device

A technician is replacing a DigiMesh device in the field and needs to have the its entry removed from a cloud server's database. The technician pushes the commissioning button on the old device once to send an identification broadcast. The server can use the broadcast to identify which device is being replaced and perform the necessary action.

When the node identification broadcast is sent, every device that receives the message will flash the association LED and emit the following information frame:

```
7E 00 27 95 00 13 A2 00 12 34 56 78 FF FE C2 FF FE 00 13 A2 00 12 34 56 78 4C
48 37 35 00 FF FE 01 01 C1 05 10 1E 00 14 00 08 0D
```

Frame type	64-bit source	Reserved	Options	64-bit remote	NI String	Reserved	Device type	Event	Profile ID	MFG ID
0x95	0x0013A200 12345678	0xFFFE	0xC2	0x0013A200 12345678	0x4C483735 00	0xFFFE	0x01	0x01	0xC105	0x101E
Identification		*Unused*	*DigiMesh broadcast*		*"LH75" + null*	*Unused*	*Router*	*Button press*	*Digi*	*Digi*

Remote AT Command Response- 0x97

Request frame: Remote AT Command Request - 0x17

Description

This frame type is emitted in response to a Remote AT Command Request - 0x17. Some commands send back multiple response frames; for example, the **ND** command. Refer to individual AT command descriptions for details on API response behavior.

This frame is only emitted if the Frame ID in the request is non-zero.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Remote AT Command Response - **0x97**
4	8-bit	**Frame ID**	Identifies the data frame for the host to correlate with a prior request.
5	64-bit	**64-bit source address**	The sender's 64-bit address.
13	16-bit	**Reserved**	Unused, but this field is typically set to **0xFFFE**.
15	16-bit	**AT command**	The two ASCII characters that identify the AT Command.
17	8-bit	**Command status**	Status code for the host's request: **0x00** = OK **0x01** = ERROR **0x02** = Invalid command **0x03** = Invalid parameter **0x04** = Transmission failure **0x0C** = Encryption error
18-n	variable	**Parameter value (optional)**	If the host requested a command parameter change, this field will be omitted. If the host queried a command by omitting the parameter value in the request, this field will return the value currently set on the device.
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Examples

Each example is written without escapes (**AP** = **1**) and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

Set remote command parameter

Host set the **NI** string of a remote device to "**Remote**" using a Remote AT Command Request - 0x17.

The corresponding 0x97 Remote AT Command Response with a matching Frame ID is emitted as a response:

```
7E 00 0F 97 27 00 13 A2 00 12 34 56 78 12 7E 4E 49 00 51
```

Frame type	Frame ID	64-bit source	Reserved	AT command	Command Status	Command data
0x97	0x27	0x0013A200 12345678	0x127E	0x4E49	0x00	(omitted)
Response	*Matches request*		*Unused*	*"NI"*	*Success*	*Parameter changes return no data*

Transmission failure

Host queued the the PAN ID change of a remote device using a Remote AT Command Request - 0x17. Due to existing network congestion, the host will retry any failed attempts.

The corresponding 0x97 Remote AT Command Response with a matching Frame ID is emitted as a response:

```
7E 00 0F 97 27 00 13 A2 00 12 34 56 78 FF FE 49 44 04 EA
```

Frame type	Frame ID	64-bit source	Reserved	AT command	Command Status	Command data
0x97	0x27	0x0013A200 12345678	0xFFFE	0x4944	0x04	(omitted)
Response	*Matches request*		*Unused*	*"ID"*	*Transmission failure*	*Parameter changes return no data*

Query remote command parameter

Query the temperature of a remote device—TP (Temperature).

The corresponding 0x97 Remote AT Command Response with a matching Frame ID is emitted with the temperature value as a response:

```
7E 00 11 97 27 00 13 A2 00 12 34 56 78 FF FE 54 50 00 00 2F A8
```

Frame type	Frame ID	64-bit source	Reserved	AT command	Command Status	Command data
0x97	0x27	0x0013A200 12345678	0x0013A200 12345678	0x4944	0x00	0x002F
Response	*Matches request*		*Unused*	*"TP"*	*Success*	*+47 °C*

Migrate from XBee through-hole to surface-mount devices

We designed the XBee surface-mount and through-hole devices to be compatible with each other and offer the same basic feature set. The surface-mount form factor has more I/O pins. Because the XBee device was originally offered in only the through-hole form factor, we offer this section to help you migrate from the through-hole to the surface-mount form factor.

Pin mapping

XBee 868LP RF Module modules are designed to be compatible with the XBee through-hole modules. The SMT modules have all the features of the through-hole modules, and offer the increased feature set.

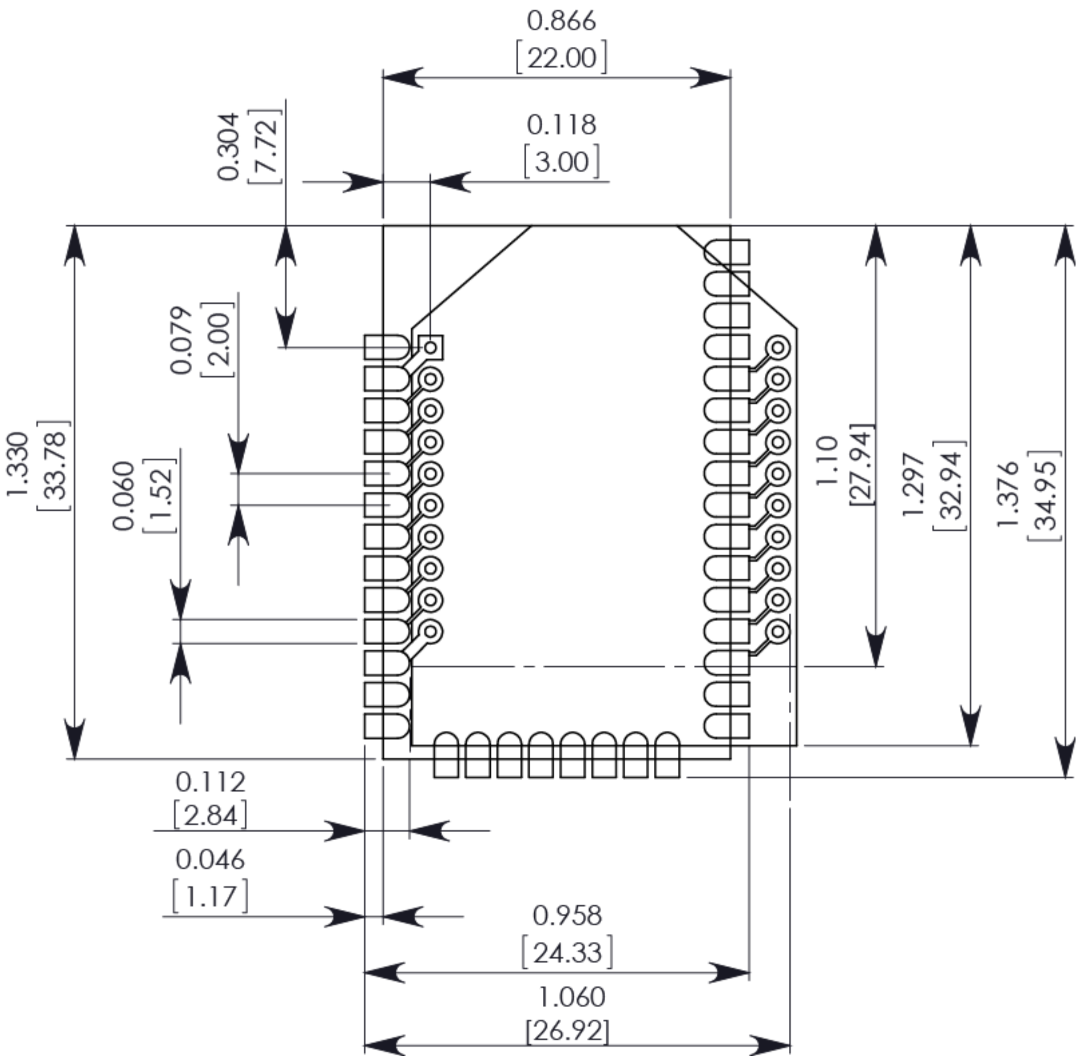
SMT Pin #	Name	TH Pin #
1	GND	
2	V_{DD}	1
3	DOUT / DIO13	2
4	DIN / $\overline{\text{CONFIG}}$ / DIO14	3
5	DIO12	4
6	$\overline{\text{RESET}}$	5
7	RSSI PWM / DIO10	6
8	PWM1 / DIO11	7
9	[reserved]	8
10	$\overline{\text{DTR}}$ / SLEEP_RQ / DIO8	9
11	GND	10
12	SPI_ATTN / DIO19	
13	GND	
14	SPI_CLK / DIO18	
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	V_{REF}	14

SMT Pin #	Name	TH 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 important difference between the surface-mount and the through-hole devices is how they mount to the PCB. Different mounting techniques are required.

We designed a footprint that allows either device to be attached to a PCB as shown in the following diagram. The dimensions without brackets are in inches, and those in brackets are in millimeters.



The round holes in the diagram are for the through-hole design, and the semi-oval pads are for the surface-mount design. Pin 1 of the through-hole design lines up with pad 1 of the surface-mount design, but the pins are actually offset by one pad—see Pin mapping. By using diagonal traces to connect the appropriate pins, the layout works for both modules.

For information on attaching the SMT device, see Manufacturing information.

Manufacturing information

The XBee 868LP RF Module is designed for surface-mounting on the OEM PCB. It has castellated pads to allow for easy solder attaching and inspection. The pads are all located on the edge of the device so there are no hidden solder joints on these devices.

Recommended solder reflow cycle

The following table lists the recommended solder reflow cycle. The chart shows the temperature setting and the time to reach the temperature.

Time (seconds)	Temperature (°C)
30	65
60	100
90	135
120	160
150	195
180	240
210	260

The maximum temperature should not exceed 260 °C.

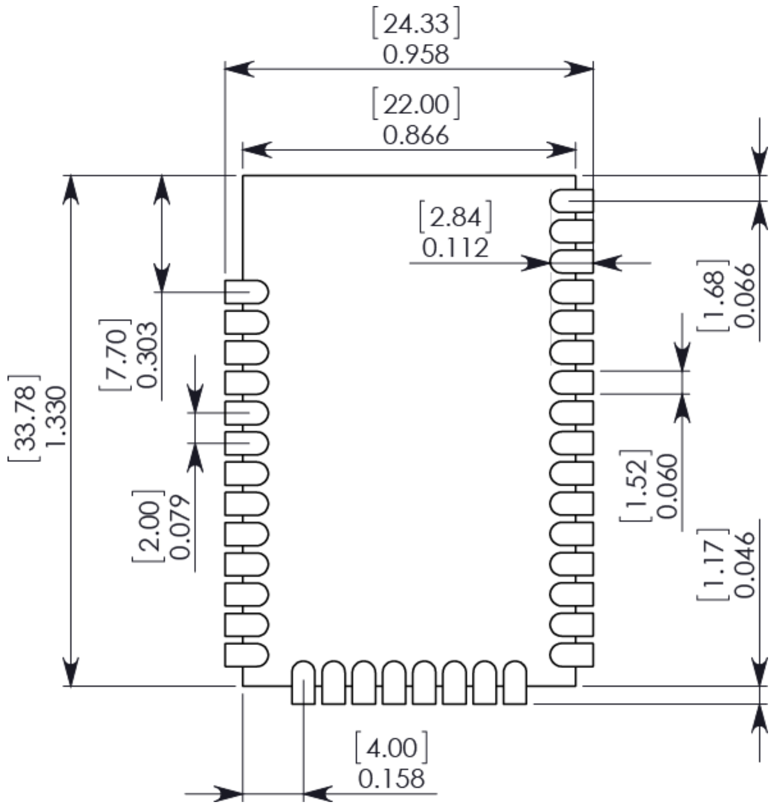
The device reflows during this cycle, and must not be reflowed upside down. Be careful not to jar the device while the solder is molten, as parts inside the device can be removed from their required locations.

Hand soldering is possible and should be done in accordance with approved standards.

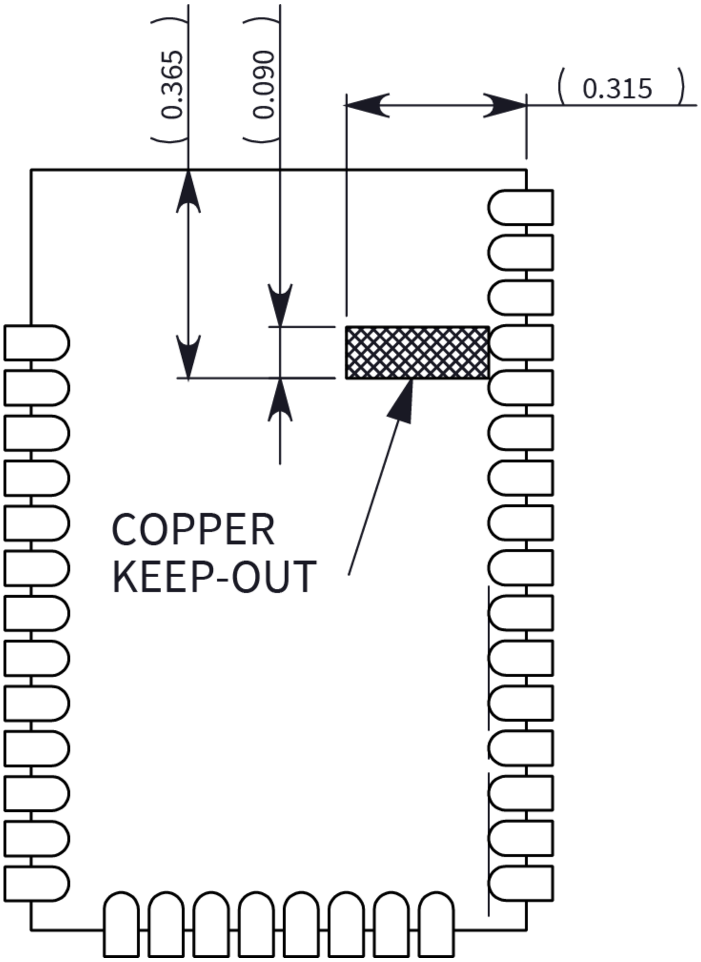
Recommended footprint and keepout

We recommend that you use the following PCB footprints for surface-mounting. The dimensions without brackets are in inches, and those in brackets are in millimeters.

Match the solder footprint 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, we recommend 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

We recommend that you use a “no clean” solder paste in assembling these devices. This eliminates the clean step and ensures that you do not leave unwanted residual flux under the device where it is difficult to remove. In addition:

- Cleaning with liquids can result in liquid remaining under the device or in the gap between the device and the host PCB. This can lead to unintended connections between pads.
- The residual moisture and flux residue under the device are not easily seen during an inspection process.

Note The best practice is to use a “no clean” solder paste to avoid the issues above and ensure proper module operation.

Reworking

Never perform rework on the device itself. The device has been optimized to give the best possible performance, and reworking the device itself will void warranty coverage and certifications. We recognize that some customers choose to rework and void the warranty. The following information serves as a guideline in such cases to increase the chances of success during rework, though the warranty is still voided.

The device may be removed from the OEM PCB by the use of a hot air rework station, or hot plate. Be careful not to overheat the device. During rework, the device temperature may rise above its internal solder melting point and care should be taken not to dislodge internal components from their intended positions.

Regulatory information

Europe

The XBee 868LP RF Modules have been tested 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 with articles 3.1a and 3.1b of the EU Directive 2014/53/EU (Radio Equipment Directive). A Declaration of Conformity must be issued for each of these standards and kept on file as described in the RE Directive (Radio Equipment 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 guide.

Maximum power and frequency specifications

The maximum radiated RF power is 14 dBm.

The following table shows channel frequencies.

Channel number	Frequency
0	863.15 MHz
1	863.35 MHz
2	863.55 MHz
3	863.75 MHz
4	863.95 MHz
5	864.15 MHz
6	864.35 MHz
7	864.55 MHz
8	864.75 MHz
9	864.95 MHz
10	865.15 MHz
11	865.35 MHz
12	865.55 MHz
13	865.75 MHz
14	865.95 MHz
15	866.15 MHz
16	866.35 MHz
17	866.55 MHz
18	866.75 MHz

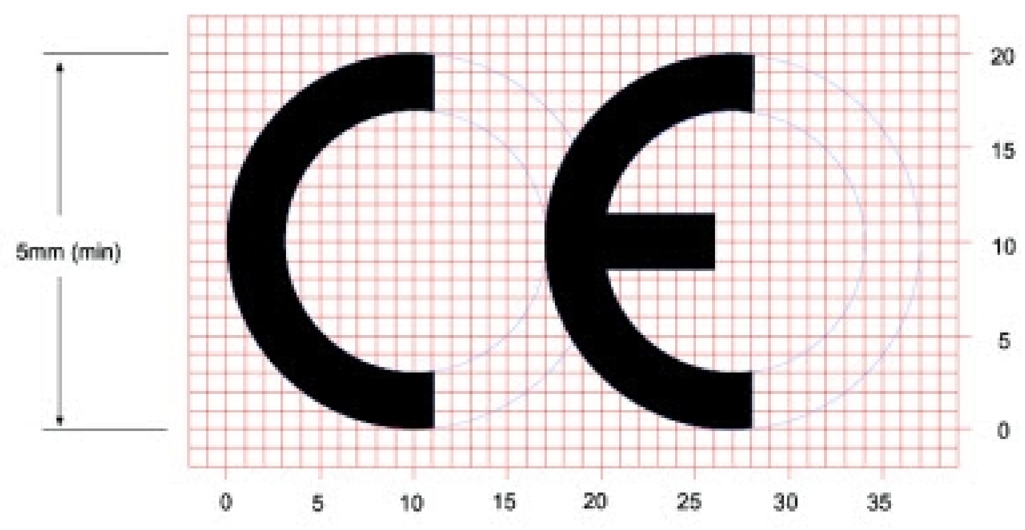
Channel number	Frequency
19	866.95 MHz
20	867.15 MHz
21	867.35 MHz
22	867.55 MHz
23	867.75 MHz
24	867.95 MHz
25	868.15 MHz
26	868.35 MHz
27	868.85 MHz
28	869.05 MHz
29	869.85 MHz

CE and UKCA OEM labeling requirements

The CE and UKCA markings must be clearly visible and legible when you affix it to the product. If this is not possible, you must attach these marks to the packaging (if any) or accompanying documents.

CE labeling requirements

The “CE” marking must be affixed to a visible location on the OEM product. The following figure shows CE labeling requirements.

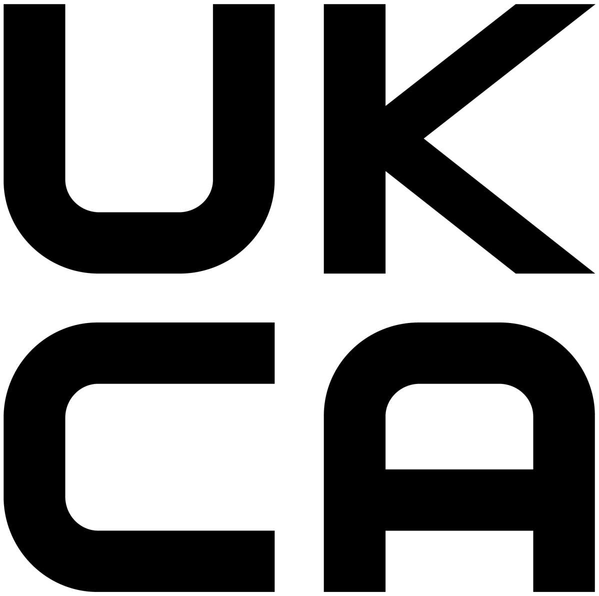


The CE mark shall consist of the initials “CE” taking the following form:

- 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 5 mm except where this is not possible on account of the nature of the apparatus.
- The CE marking must be affixed visibly, legibly, and indelibly.

UK Conformity Assessed (UKCA) labeling requirements



See guidance/using-the-ukca-marking for further details.

You must make sure that:

- if you reduce or enlarge the size of your marking, the letters forming the UKCA marking must be in proportion to the version set out below
- the UKCA marking is at least 5 mm in height – unless a different minimum dimension is specified in the relevant legislation
- the UKCA marking is easily visible, legible (from 1 January 2023 it must be permanently attached)
- the UKCA marking can take different forms (for example, the colour does not have to be solid), as long as it remains visible, legible and maintains the required proportions.

Important note

Digi customers assume full responsibility for learning and meeting the required guidelines for each country in their distribution market. Refer to the radio regulatory agency in the desired countries of operation for more information.

Declarations of conformity

Digi has issued Declarations of Conformity for the XBee RF Modules concerning emissions, EMC, and safety. For more information, see www.digi.com/resources/certifications.

Antennas

The following antennas have been tested and approved for use with the XBee 868LP RF Module:

All antenna part numbers followed by an asterisk (*) are not available from Digi. Consult with an antenna manufacturer for an equivalent option.

- Dipole (2.1 dBi), Digi PN A08-HABUF-P5I*
- PCB Antenna (-9 dBi), included with the module