Digi XBee® 3 802.15.4

Radio Frequency (RF) Module

User Guide

Revision history—90002273

Revision	Date	Description
D	August 2019	Added **%P** and **DM**. Updated **RR**. Updates to Remote AT Command Request frame. Added location and BLE commands. Added statuses to the 0x8A frame. Added frames Ox2C, 0x2D, 0xAC, and 0xAD. Added Get started with BLE and BLE reference sections. Made changes to the CCA operations section.
E	September 2019	Added ***S**, ***V**, ***W**, ***X**, ***Y**, **SA**, **AZ**, and **FS INFO FULL**. Added reserved endpoints to 0x11 frame.
F	April 2020	Added **BP**, **R?**, **US**. Updated OTA firmware/file system upgrades.
G	May 2020	Revised all API frame descriptions.
H	August 2020	Added a note to **D8**. Updated **OTA firmware/file system upgrades**.

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 (Digi XBee® 3 802.15.4 RF Module User Guide, 90002273 H) in the subject line of your email.

Contents

Secure access

File system

Get started with BLE

BLE reference

Configure the XBee 3 802.15.4 RF Module

Modes

Serial communication

SPI operation

I/O support

Networking

Network commissioning and diagnostics

Sleep support

AT commands

Operate in API mode

Frame descriptions

OTA firmware/file system upgrades

OTA file system upgrades

Digi XBee® 3 802.15.4 RF Module User Guide

XBee 3 802.15.4 RF Modules are embedded solutions providing wireless end-point connectivity to devices. These devices use the IEEE 802.15.4 networking protocol for fast point-to-multipoint or peer-to-peer networking. They are designed for high-throughput applications requiring low latency and predictable communication timing.

The XBee 3 802.15.4 RF Module supports the needs of low-cost, low-power wireless sensor networks. The devices require minimal power and provide reliable delivery of data between devices. The devices operate within the ISM 2.4 GHz frequency band.

The XBee 3 802.15.4 RF Module uses XBee 3 hardware and the Silicon Labs EFR32 chipset. As the name suggests, the 802.15.4 module is over-the-air compatible with our Legacy 802.15.4 modules (S1 and S2C hardware).

For information about XBee 3 hardware, see the XBee 3 RF Module Hardware Reference Manual.

Applicable firmware and hardware

This manual supports the following firmware:

- v.20xx Digi 802.15.4

It supports the following hardware:

- XBee 3

Change the firmware protocol

You can switch the firmware loaded onto the XBee 3 hardware to run any of the following protocols:

- Zigbee
- 802.15.4
- DigiMesh

To change protocols, use the **Update firmware** feature in XCTU and select the firmware. See the *XCTU User Guide*.

Regulatory information

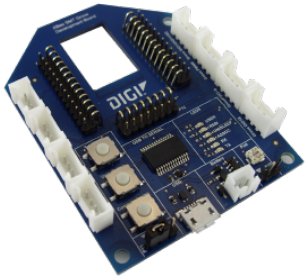
See the Regulatory information section of the *XBee 3 RF Module Hardware Reference Manual* for the XBee 3 hardware's regulatory and certification information.

Get started

This section covers the following tasks and features:

Verify kit contents

The XBee 3 802.15.4 RF Module development kit contains the following components:

Part	
XBee 3 Zigbee SMT module (3)	
XBee Grove development board (3)	
Micro USB cable (3)	
Antenna - 2.4 GHz, half-wave dipole, 2.1 dBi, U.FL female, articulating (3)	
XBee stickers	

Assemble the hardware

This guide walks you through the steps required to assemble and disassemble the hardware components of your kit.

- Plug in the XBee 3 802.15.4 RF Module
- Unplug an XBee 3 802.15.4 RF Module

The kit includes several XBee Grove Development Boards. For more information about this hardware, see the XBee Grove Development Board documentation.

Plug in the XBee 3 802.15.4 RF Module

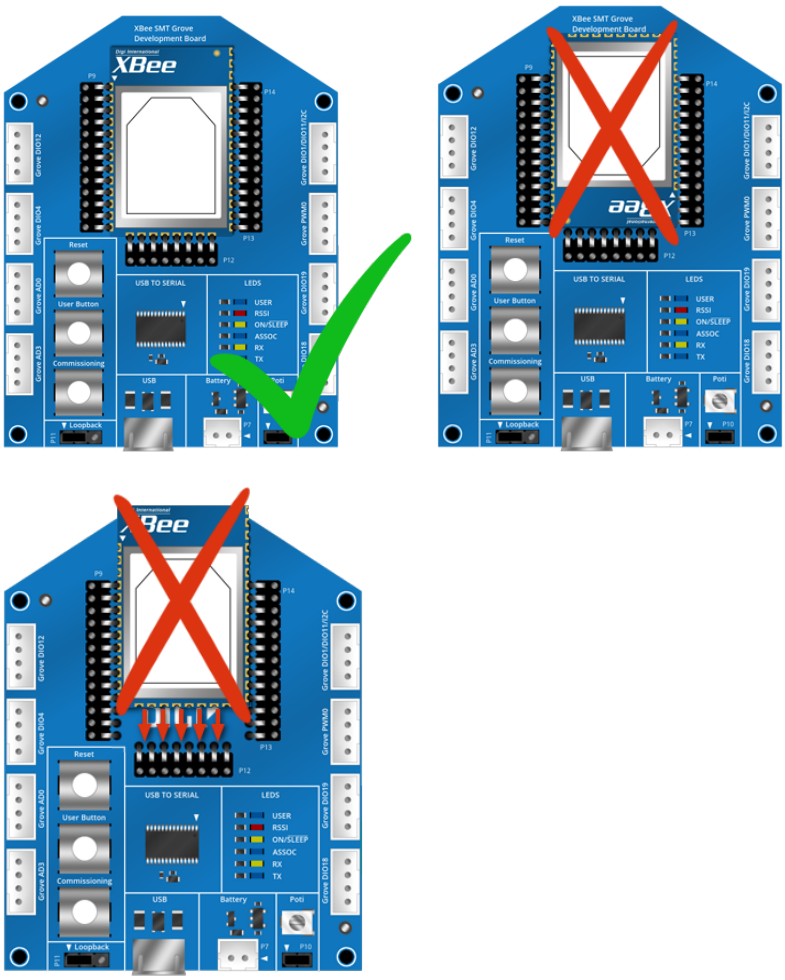
This kit includes two XBee Grove Development Boards. For more information about this hardware, visit the XBee Grove Development Board documentation.

Follow these steps to connect the XBee devices to the boards included in the kit:

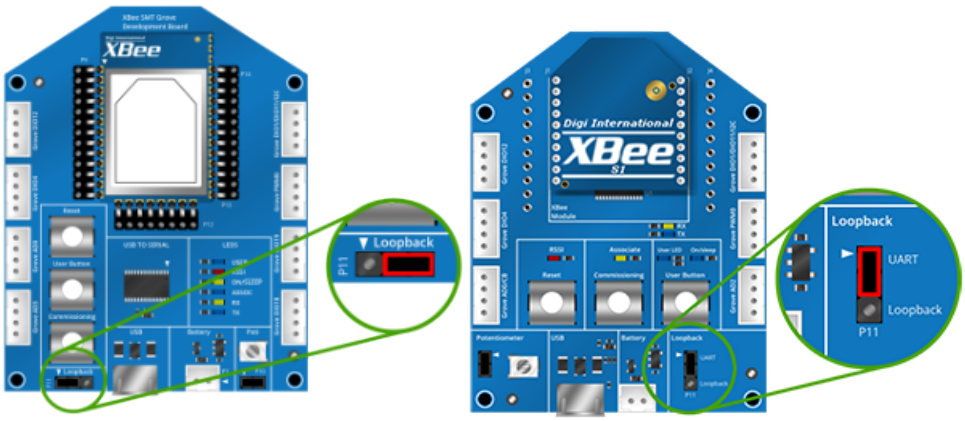
1. Plug one XBee 3 802.15.4 RF Module into each XBee Grove Development Board. When you connect the development board to a PC for the first time, the PC automatically installs drivers, which may take a few minutes to complete.

CAUTION! Never insert or remove the XBee while the power is on (either from the micro USB or a battery)!

For XBee SMT devices, align all XBee pins with the spring header and carefully push the device until it clicks firmly into the board.



2. Once theXBee 3 802.15.4 RF Module is plugged into the board, connect the board to your computer using the micro USB cables provided.
3. Ensure the loopback jumper is in the UART position.



Unplug an XBee 3 802.15.4 RF Module

To disconnect a device from the XBee Grove Development Board:

1. Disconnect the micro USB cable from the board so it is not powered.
2. Remove the device from the board socket, taking care not to bend any of the pins. The surface mount device uses spring pins rather than a socket and has a rectangular board cutout designed to help in removing the XBee 3 802.15.4 RF Module.

CAUTION! Make sure the board is **not** powered when you remove the XBee 3 802.15.4 RF Module.

Configure the device using XCTU

XBee Configuration and Test Utility (XCTU) is a multi-platform program that enables users to interact with Digi radio frequency (RF) devices through a graphical interface. The application includes built-in tools that make it easy to set up, configure, and test Digi RF devices.

For instructions on downloading and using XCTU, see the *XCTU User Guide*.

Configure remote devices

You can communicate with remote devices over the air through a corresponding local device.

Note Using API mode on the local device allows you to send remote API commands.

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.
2. Load XBee 3 802.15.4 firmware onto each device if it is not already loaded. See How to update

the firmware of your modules in the *XCTU User Guide* for more information.

3. Configure the first device in API mode and name it **XBEE_A** by configuring the following parameters:

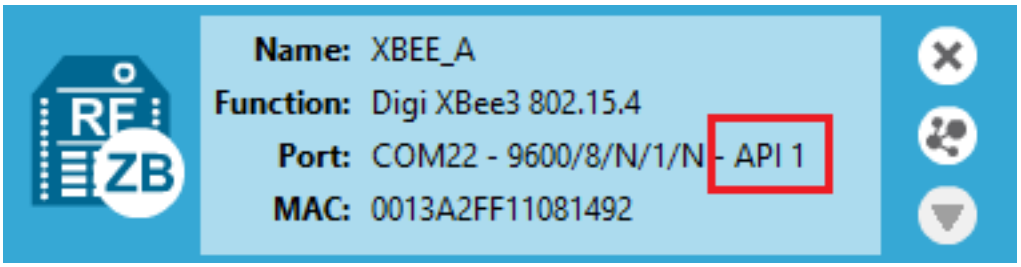
- **ID**: 2018
- **NI**: XBEE_A
- **AP**: API enabled [1]

4. Configure the second device in either API or Transparent mode, and name it **XBEE_B** by configuring the following parameters:

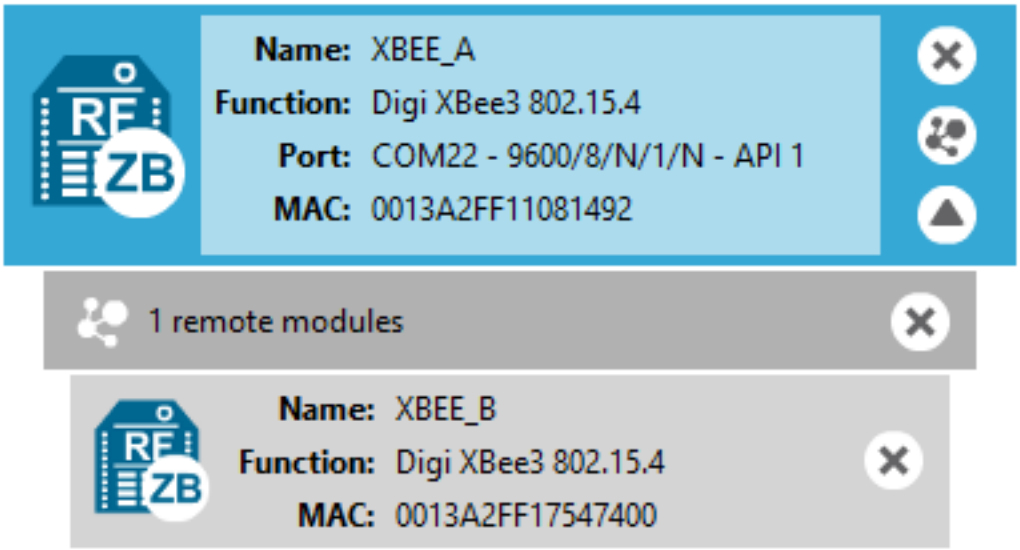
- **ID**: 2018
- **NI**: XBEE_B
- **AP**: 0 or 1

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.



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.



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

   LT: FF (hexadecimal representation for 2550 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**.

Configure the devices for a range test

1. Add two devices to XCTU.
2. Select the first module and click the **Load default firmware settings** button.
3. Configure the following parameters:

   ID: 2018
   NI: LOCAL_DEVICE
   AP: API Mode Enabled [1]

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

   ID: 2018

   NI: REMOTE_DEVICE

   AP: Transparent mode [0] (The remote node must be in transparent mode to loop back packets)

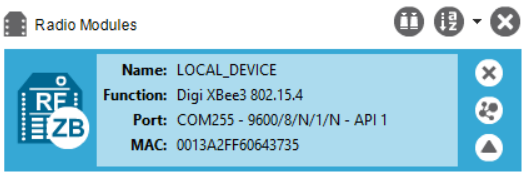
7. Click the **Write radio settings** button.

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

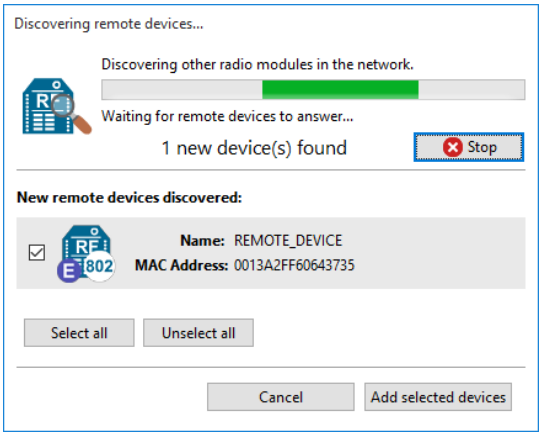
8. Disconnect REMOTE_DEVICE from the computer, remove it from XCTU, and connect it to a power supply, laptop, or portable battery.
9. Leave LOCAL_DEVICE connected to the computer.

Perform a range test

1. Go to the XCTU display for radio 1.

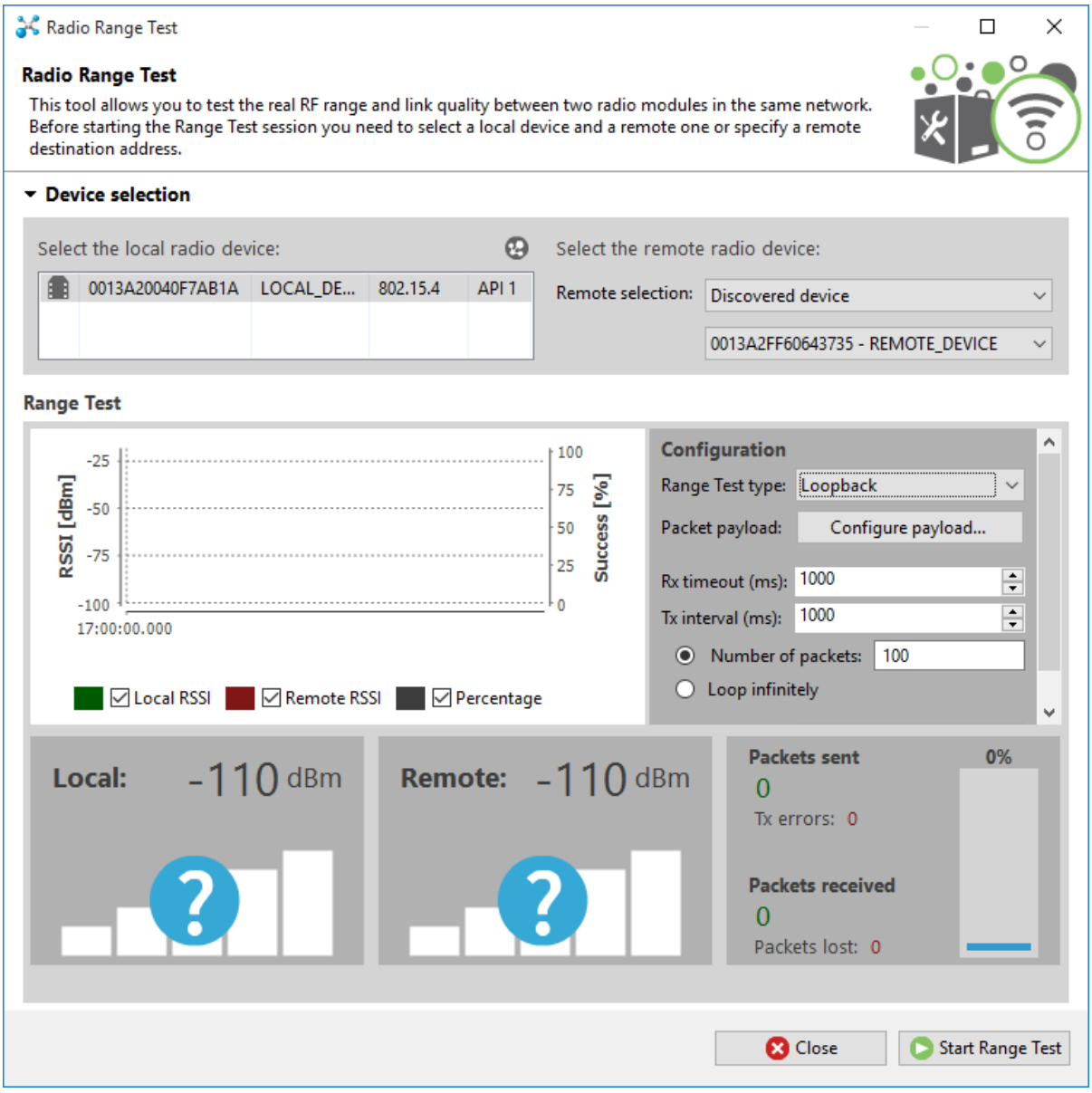


2. Click to discover remote devices within the same network. The **Discover remote devices** dialog appears.

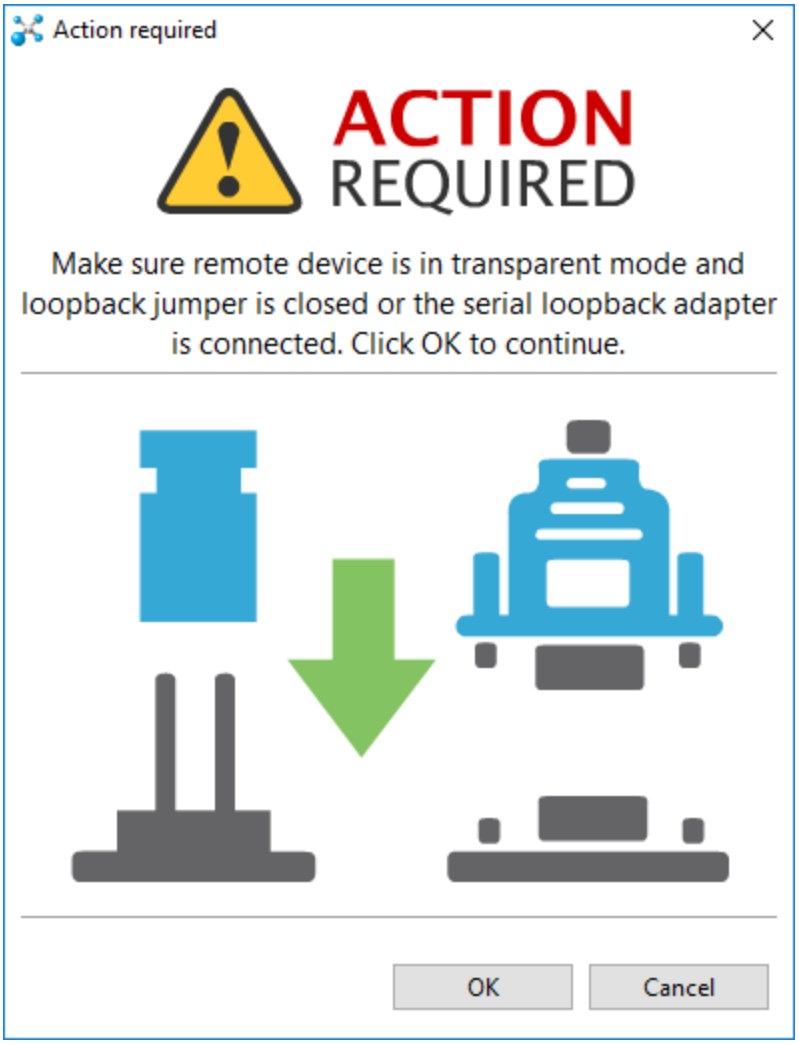


3. Click **Add selected devices**.

4. Click and select **Range test**. The **Radio Range Test** dialog appears.

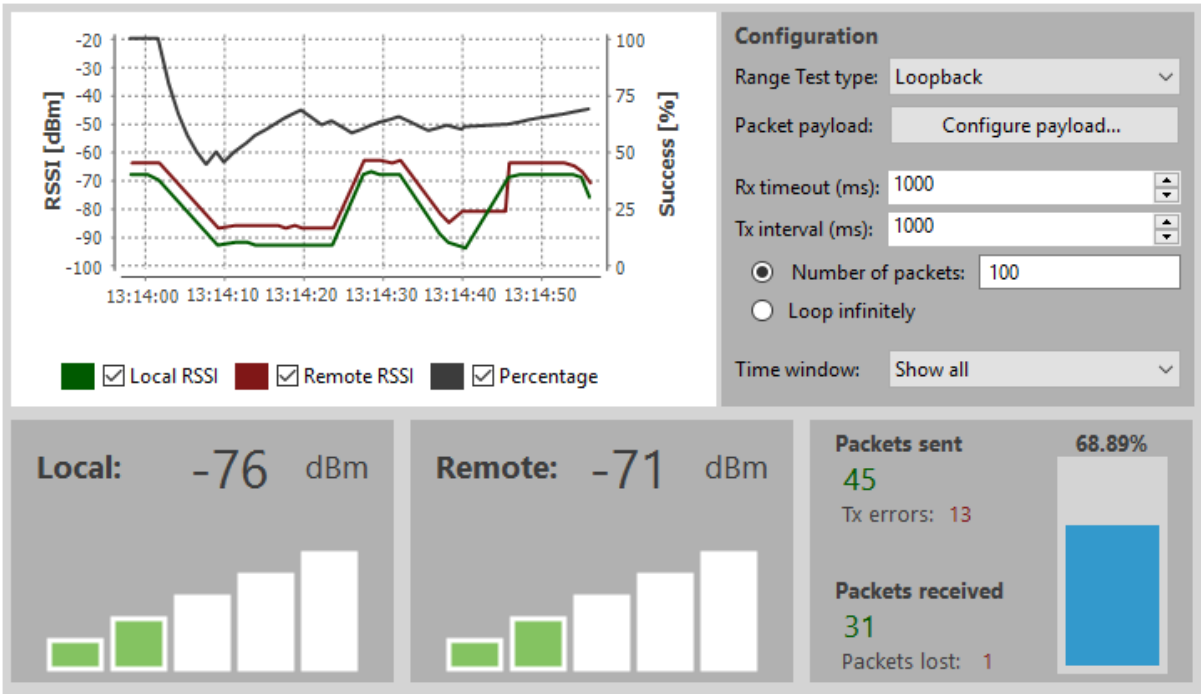


5. Change the **Range Test type** to **Loopback**.
6. In the **Select the local radio device** area, select radio 1. XCTU automatically selects the **Discovered device** option, and the **Start Range Test** button is active.
7. Click Start Range Test to begin the range test. XCTU prompts you to enable the loopback jumper.

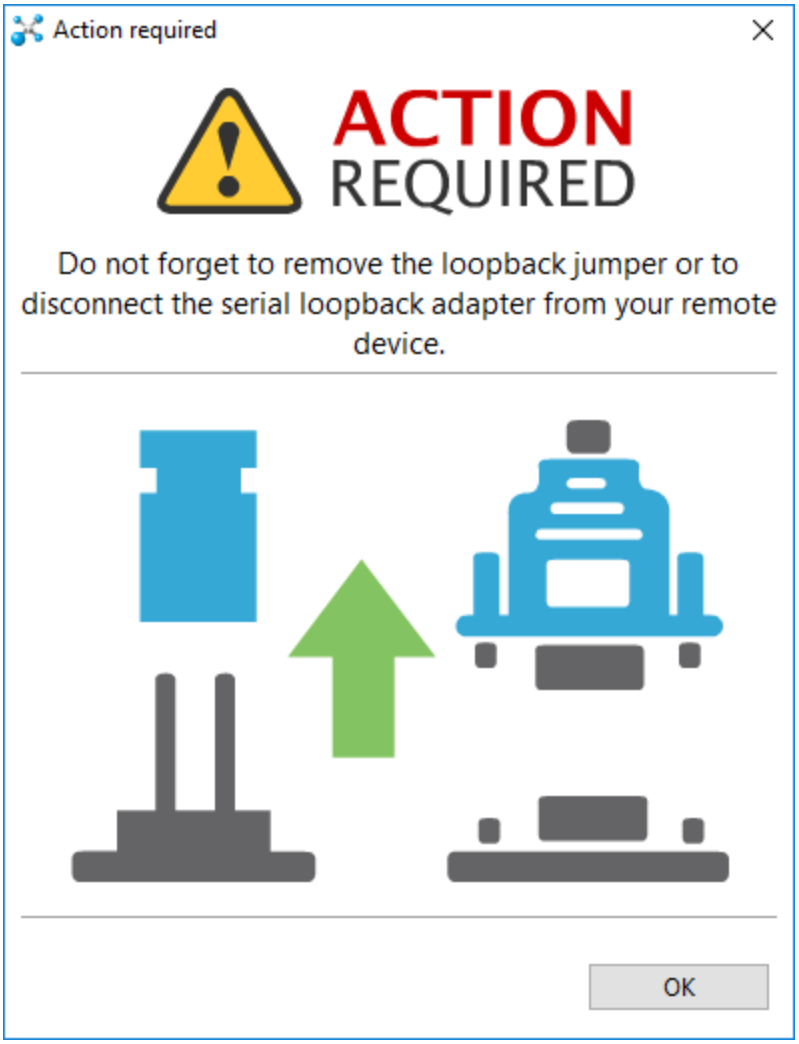


Plug in the XBee 3 802.15.4 RF Module has pictures that show the jumper in the UART position—move the jumper to the left on the surface-mount device or down on the through-hole device puts it in loopback mode

If the test is running properly, the packets sent should match the packets received. You will also see the received signal strength indicator (RSSI) update for each radio after each reception.



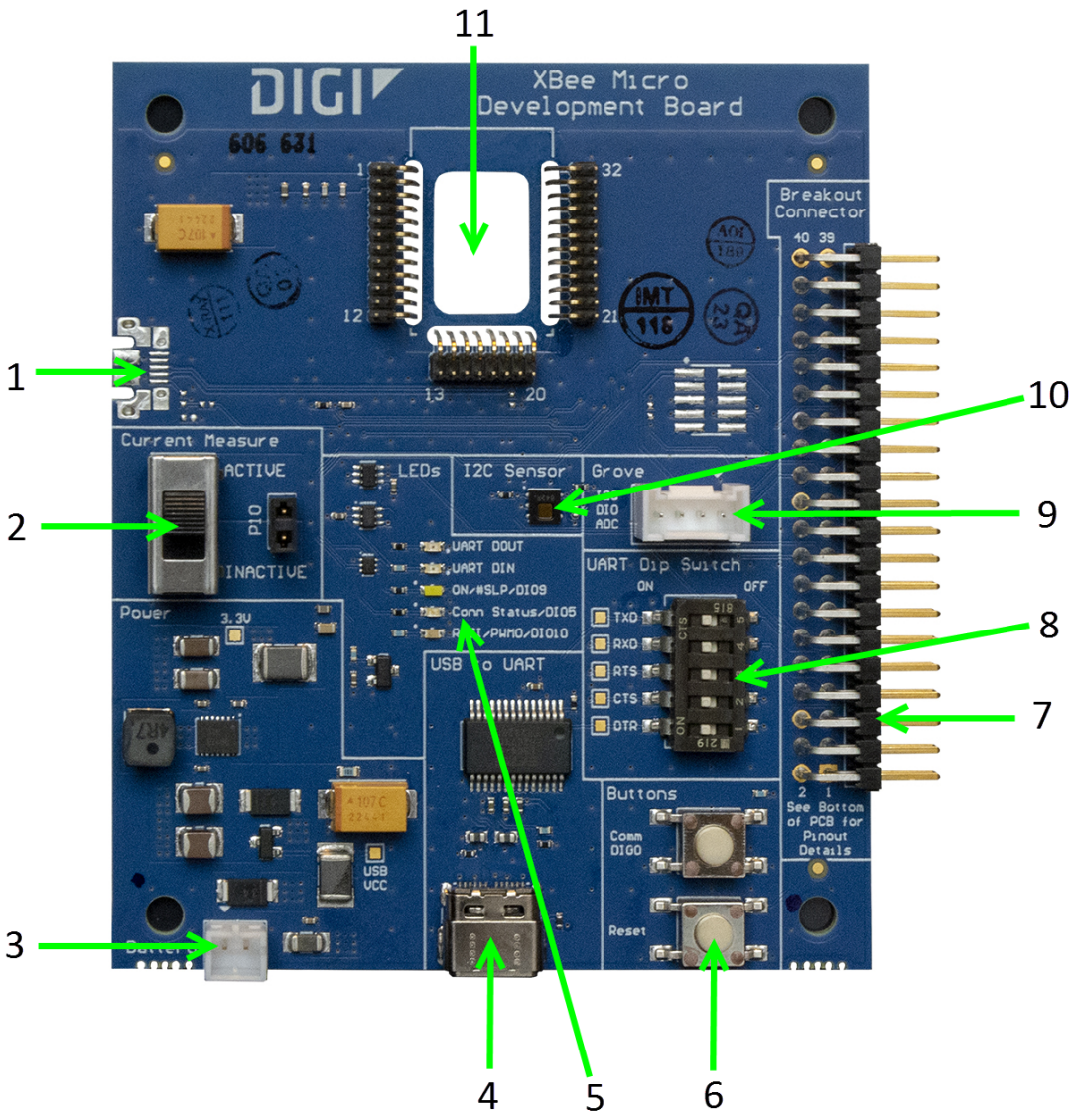
8. Move Radio 1 around to see the resulting signal strength at different distances. You can also test different data rates by reconfiguring the **BR** (data rate) parameter on both radios. When the test is complete, click **Stop Range Test**. XCTU displays another loopback jumper warning screen reminding you to put the loopback jumper back in its original position.



XBIB-C Micro Mount reference

This picture shows the XBee-C Micro Mount development board and the table that follows explains the callouts in the picture.

Note This board is sold separately.

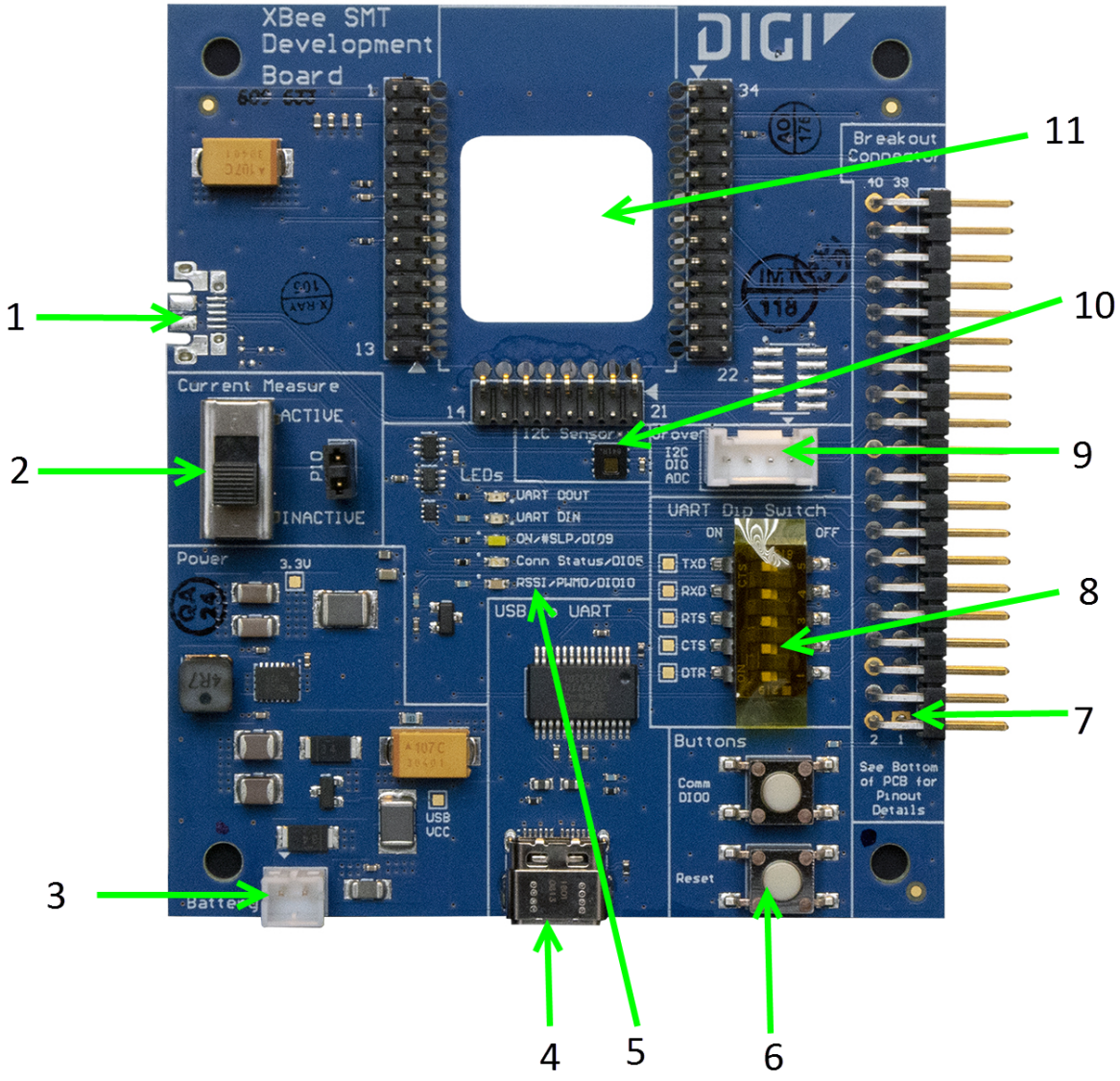

11
1
2
3
4
5
6
7
8
9
10
DIGI
XBee Micro
Development Board
606 631
Breakout
Connector
40 39
2 1
See Bottom
of PCB for
Pinout
Details
Current Measure
ACTIVE
INACTIVE
PIO
LEDs
I2C Sensor
Grove
DIO
ADC
UART DOUT
UART DIN
ON/#SLP/DIO9
Conn Status/DIO5
RSSI/PWM0/DIO10
UART Dip Switch
ON
OFF
TXD
RXD
RTS
CTS
DTR
Power
3.3V
USB
VCC
Buttons
Comm
DIO0
Reset

Number	Item	Description
1	Secondary USB (USB MICRO B)	Secondary USB Connector for possible future use. Not used.
2	Current Measure	Large switch controls whether current measure mode is active or inactive. When inactive, current can freely flow to the VCC pin of the XBee. When active, the VCC pin of the XBee is disconnected from the 3.3 V line on the development board. This allows current measurement to be conducted by attaching a current meter across the jumper P10.
3	Battery Connector	If desired, you can attach a battery to provide power to the development board. The voltage can range from 2 V to 5 V. The positive terminal is on the left.
4	USB-C Connector	Connects to your computer. This is connected to a USB to UART conversion chip that has the five UART lines passed to the XBee device. The UART Dip Switch can be used to disconnect these UART lines from the XBee.
5	LED indicator	Red: UART DOUT (modem sending serial/UART data to host) Green: UART DIN (modem receiving serial/UART data from host) White: ON/SLP/DIO9 Blue: Connection Status/DIO5 Yellow: RSSI/PWM0/DIO10
6	User Buttons	Comm DIO0 Button connects the Commissioning/DIO0 pin on the XBee Connector through to a 10 Ω resistor to GND when pressed. $\overline{\text{RESET}}$ Button Connects to the $\overline{\text{RESET}}$ pin on the XBee Connector to GND when pressed.
7	Breakout Connector	This 40-pin connector can be used to connect to various XBee pins as shown on the silkscreen on the bottom of the board.
8	UART Dip Switch	This dip switch allows the user to disconnect any of the primary UART lines on the XBee from the USB to UART conversion chip. This allows for testing on the primary UART lines without the USB to UART conversion chip interfering. Push Dip switches to the right to disconnect the USB to UART conversion chip from the XBee.
9	Grove Connector	This connector can be used to attach I2C enabled devices to the development board. Note that I2C needs to be available on the XBee in the board to use this functionality. Pin 1: I2C_CLK/XBee DIO1 Pin2: I2C_SDA/XBee DIO11 Pin3: VCC Pin4: GND
10	Temp/Humidity Sensor	This as a Texas Instruments HDC1080 temperature and humidity sensor. This part is accessible through I2C. Be sure that the XBee that is inserted into the development board has I2C if access to this sensor is desired.
11	XBee Socket	This is the socket for the XBee (Micro form factor).

XBIB-C SMT reference

This picture shows the XBee-C SMT development board and the table that follows explains the callouts in the picture.

Note This board is sold separately.

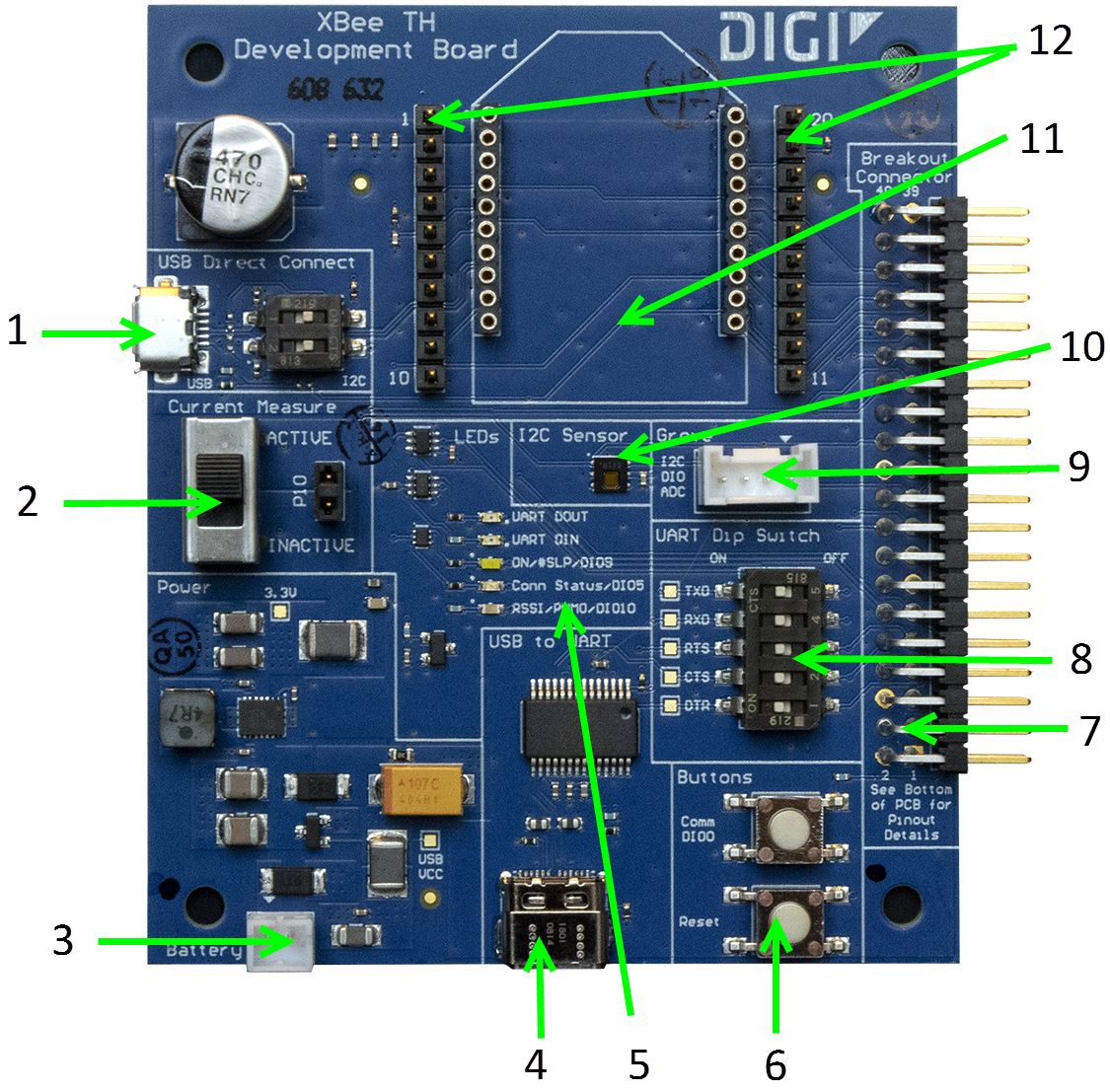


Number	Item	Description
1	Secondary USB (USB MICRO B)	Secondary USB Connector for possible future use. Not used.
2	Current Measure	Large switch controls whether current measure mode is active or inactive. When inactive, current can freely flow to the VCC pin of the XBee. When active, the VCC pin of the XBee is disconnected from the 3.3 V line on the dev board. This allows current measurement to be conducted by attaching a current meter across the jumper P10.
3	Battery Connector	If desired, you can attach a battery to provide power to the development board. The voltage can range from 2 V to 5 V. The positive terminal is on the left.
4	USB-C Connector	Connects to your computer. This is connected to a USB to UART conversion chip that has the five UART lines passed to the XBee. The UART Dip Switch can be used to disconnect these UART lines from the XBee.
5	LED indicator	Red: UART DOUT (modem sending serial/UART data to host) Green: UART DIN (modem receiving serial/UART data from host) White: ON/SLP/DIO9 Blue: Connection Status/DIO5 Yellow: RSSI/PWM0/DIO10
6	User Buttons	Comm DIO0 Button connects the Commissioning/DIO0 pin on the XBee Connector through to a 10 Ω resistor to GND when pressed. $\overline{\text{RESET}}$ Button Connects to the $\overline{\text{RESET}}$ pin on the XBee Connector to GND when pressed.
7	Breakout Connector	This 40-pin connector can be used to connect to various XBee pins as shown on the silkscreen on the bottom of the board.
8	UART Dip Switch	This dip switch allows the user to disconnect any of the primary UART lines on the XBee from the USB to UART conversion chip. This allows for testing on the primary UART lines without the USB to UART conversion chip interfering. Push Dip switches to the right to disconnect the USB to UART conversion chip from the XBee.
9	Grove Connector	This connector can be used to attach I2C enabled devices to the development board. Note that I2C needs to be available on the XBee in the board to use this functionality. Pin 1: I2C_CLK/XBee DIO1 Pin2: I2C_SDA/XBee DIO11 Pin3: VCC Pin4: GND
10	Temp/Humidity Sensor	This as a Texas Instruments HDC1080 temperature and humidity sensor. This part is accessible through I2C. Be sure that the XBee that is inserted into the Dev Board has I2C if access to this sensor is desired.
11	XBee Socket	This is the socket for the XBee (SMT form factor)

XBIB-CU TH reference

This picture shows the XBee-CU TH development board and the table that follows explains the callouts in the picture.

Note This board is sold separately.



Number	Item	Description
1	Secondary USB (USB MICRO B) and DIP Switch	Secondary USB Connector for direct programming of modules on some XBee units. Flip the Dip switches to the right for I2C access to the board; flip Dip switches to the left to disable I2C access to the board. The USB_P and USB_N lines are always connected to the XBee, regardless of Dip switch setting. This USB port is not designed to power the module or the board. Do not plug in a USB cable here unless the board is already being powered through the main USB-C connector. Do not attach a USB cable here if the Dip switches are pushed to the right.
		 **WARNING!** Direct input of USB lines into XBee units or I2C lines not designed to handle 5V can result in the destruction of the XBee or I2C components. Could cause fire or serious injury. Do not plug in a USB cable here if the XBee device is not designed for it and do not plug in a USB cable here if the Dip switches are pushed to the right.
2	Current Measure	Large switch controls whether current measure mode is active or inactive. When inactive, current can freely flow to the VCC pin of the XBee. When active, the VCC pin of the XBee is disconnected from the 3.3 V line on the development board. This allows current measurement to be conducted by attaching a current meter across the jumper P10.
3	Battery Connector	If desired, a battery can be attached to provide power to the development board. The voltage can range from 2 V to 5 V. The positive terminal is on the left. If the USB-C connector is connected to a computer, the power will be provided through the USB-C connector and not the battery connector.
4	USB-C Connector	Connects to your computer and provides the power for the development board. This is connected to a USB to UART conversion chip that has the five UART lines passed to the XBee. The UART Dip Switch can be used to disconnect these UART lines from the XBee.
5	LED indicator	Red: UART DOUT (modem sending serial/UART data to host) Green: UART DIN (modem receiving serial/UART data from host) White: ON/SLP/DIO9 Blue: Connection Status/DIO5 Yellow: RSSI/PWM0/DIO10
6	User Buttons	Comm DIO0 Button connects the Commissioning/DIO0 pin on the XBee Connector through to a 10 Ω resistor to GND when pressed. $\overline{\text{RESET}}$ Button Connects to the $\overline{\text{RESET}}$ pin on the XBee Connector to GND when pressed.
7	Breakout Connector	This 40 pin connector can be used to connect to various XBee pins as shown on the silkscreen on the bottom of the board.

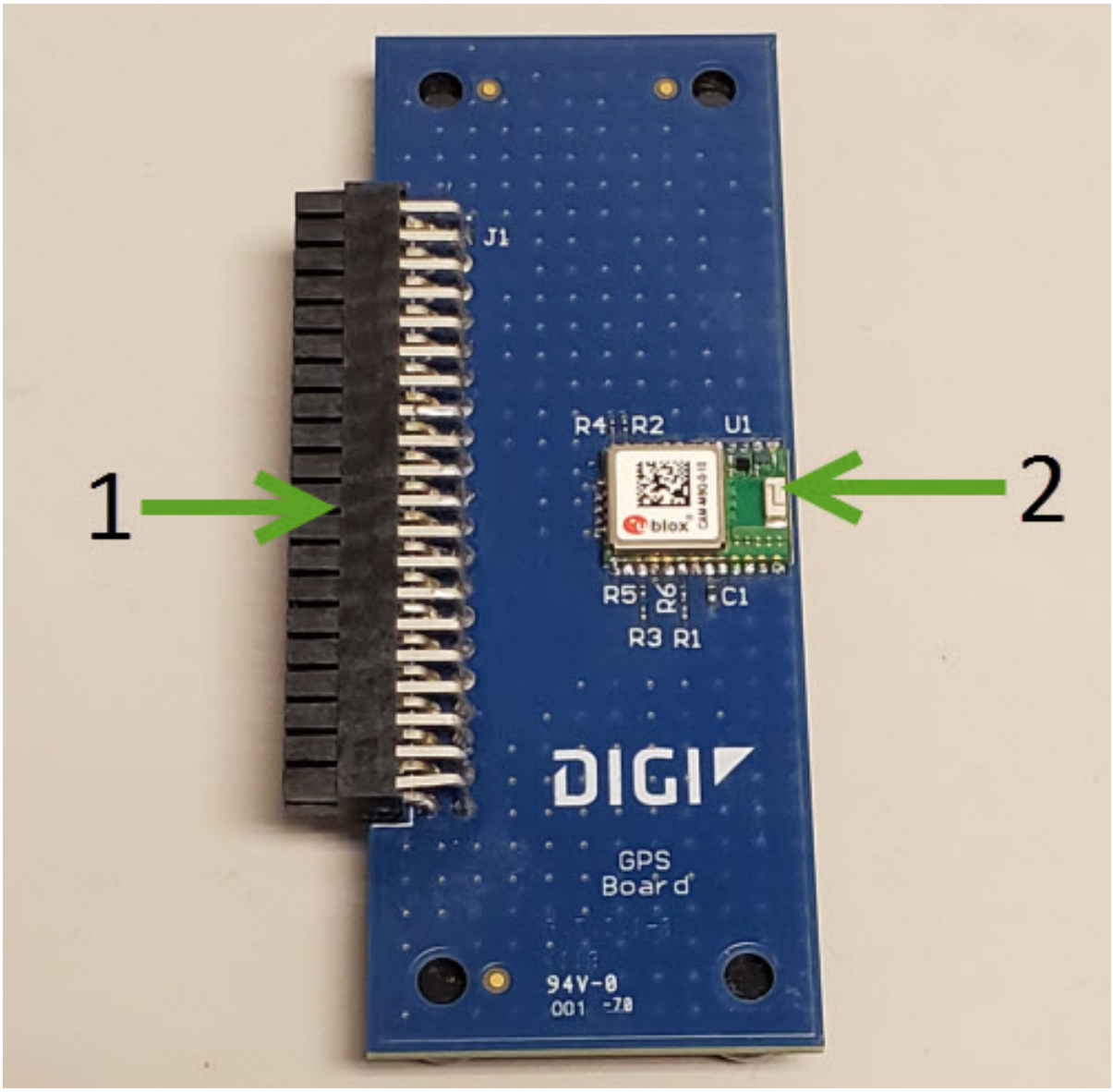
Number	Item	Description
8	UART Dip Switch	This dip switch allows the user to disconnect any of the primary UART lines on the XBee from the USB to UART conversion chip. This allows for testing on the primary UART lines without the USB to UART conversion chip interfering. Push Dip switches to the right to disconnect the USB to UART conversion chip from the XBee.
9	Grove Connector	This connector can be used to attach I2C enabled devices to the development board. Note that I2C needs to be available on the XBee in the board for this functionality to be used. Pin 1: I2C_CLK/XBee DIO1 Pin2: I2C_SDA/XBee DIO11 Pin3: VCC Pin4: GND
10	Temp/Humidity Sensor	This as a Texas Instruments HDC1080 temperature and humidity sensor. This part is accessible through I2C. Be sure that the XBee that is inserted into the development board has I2C if access to this sensor is desired.
11	XBee Socket	This is the socket for the XBee (TH form factor).
12	XBee Test Point Pins	Allows easy access for probes for all 20 XBee TH pins. Pin 1 is shorted to Pin 1 on the XBee and so on.

XBIB-C-GPS reference

This picture shows the XBIB-C-GPS module and the table that follows explains the callouts in the picture.

Note This board is sold separately. You must also have purchased an XBIB-C through-hole, surface-mount, or micro-mount development board.

Note For a demonstration of how to use MicroPython to parse some of the GPS NMEA sentences from the UART, print them and report them to Digi Remote Manager, see Run the MicroPython GPS demo.

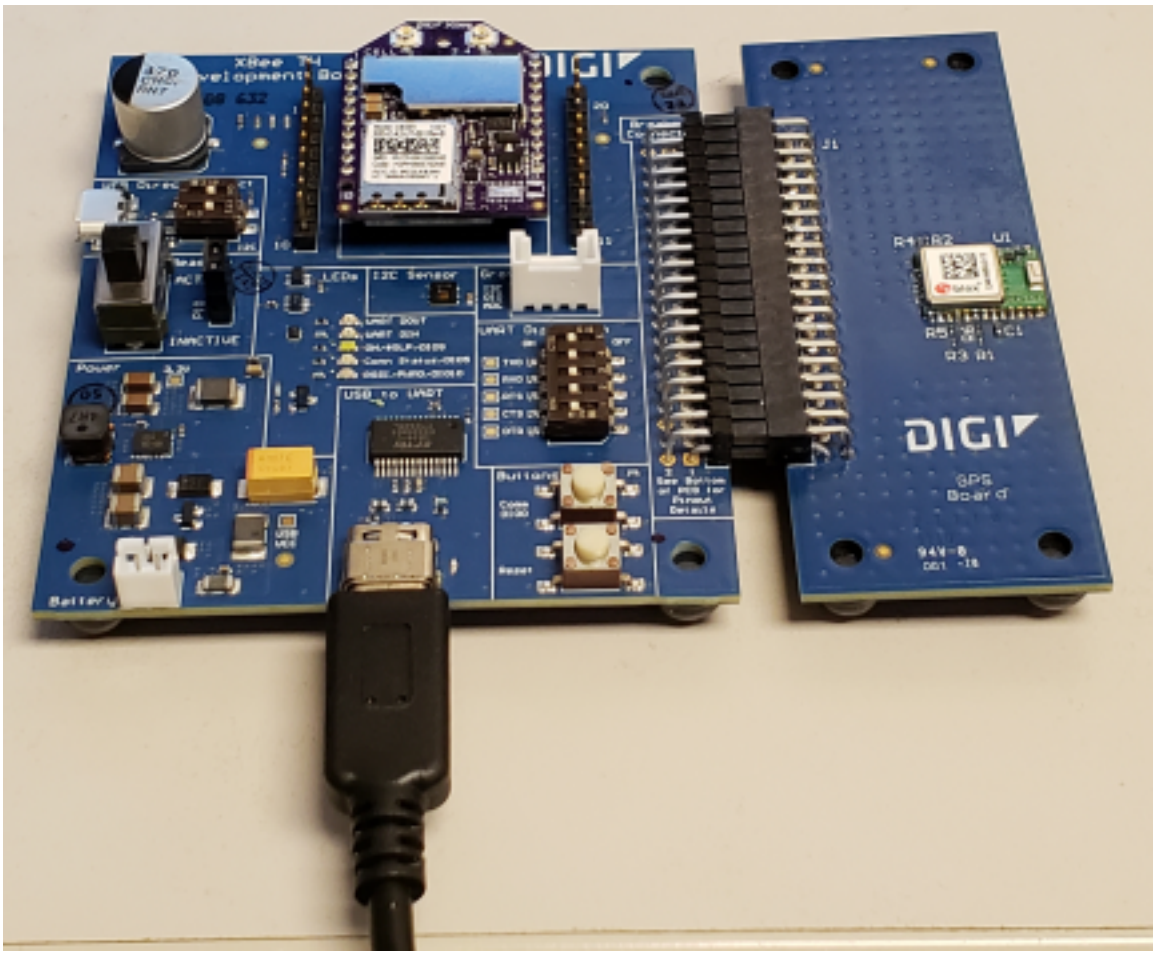

1
2
J1
R4
R2
U1
R5
R6
C1
R3
R1
DIGI
GPS
Board
94V-0

Number	Item	Description
1	40-pin header	This header is used to connect the XBIB-C-GPS board to a compatible XBIB development board. Insert the XBIB-C-GPS module slowly with alternating pressure on the upper and lower parts of the connector. If added or removed improperly, the pins on the attached board could bend out of shape.
2	GPS unit	This is the CAM-M8Q-0-10 module made by u-blox. This is what makes the GPS measurements. Proper orientation is with the board laying completely flat, with the module facing towards the sky.

Interface with the XBIB-C-GPS module

The XBee 3 802.15.4 RF Module can interface with the XBIB-C-GPS board through the large 40-pin header. This header is designed to fit into XBIB-C development board. This allows the XBee 3 802.15.4 RF Module in the XBIB-C board to communicate with the XBIB-C-GPS board—provided the XBee device used has MicroPython capabilities (see this link to determine which devices have MicroPython capabilities). There are two ways to interface with the XBIB-C-GPS board: through the host board's Secondary UART or through the I2C compliant lines.

The following picture shows a typical setup:



I²C communication

There are two I2C lines connected to the host board through the 40-pin header, SCL and SDA. I2C communication is performed over an I2C-compliant Display Data Channel. The XBIB-C-GPS module operates in slave mode. The maximum frequency of the SCL line is 400 kHz. To access data through the I2C lines, the data must be queried by the connected XBee 3 802.15.4 RF Module.

For more information about I2C Operation see the **I2C** section of the *Digi Micro Python Programming Guide*.

For more information on the operation of the XBIB-C-GPS board see the CAM-M8 datasheet. Other CAM-M8 documentation is located here.

UART communication

There are two UART pins connected from the XBIB-C-GPS to the host board by the 40-pin header: RX and TX. By default, the UART on the XBIB-C-GPS board is active and sends GPS readings to the connected device's secondary UART pins. Readings are transmitted once every second. The baud rate of the UART is 9600 baud.

For more information about using Micro Python to communicate to the XBIB-C-GPS module, see Class UART.

Run the MicroPython GPS demo

The Digi MicroPython github repository contains a GPS demo program that parses some of the GPS NMEA sentences from the UART, prints them and also reports them to Digi Remote Manager.

Note If you are unfamiliar with MicroPython on XBee you should first run some of the tutorials earlier in this manual to familiarize yourself with the environment. See Get started with MicroPython. For more detailed information, refer to the *Digi MicroPython Programming Guide*.

Step 1: Create a Remote Manager developer account

You must have a Remote Manager developer account to be able to use this program. Make sure you know the user name and password for this account.

If you don't currently have a Remote Manager developer account, you can create a free developer account.

Step 2: Download or clone the XBee MicroPython repository

1. Navigate to: **https://github.com/digidotcom/xbee-micropython/**
2. Click **Clone or download**.
3. You must either clone or download a zip file of the repository. You can use either method.
   - **Clone**: If you are familiar with GIT, follow the standard GIT process to clone the repository.
   - **Download**
     a. Click **Download zip** to download a zip file of the repository to the download folder of your choosing.
     b. Extract the repository to a location of your choosing on your hard drive.

Step 3: Edit the MicroPython file

1. Navigate to the location of the repository zip file that you created in Step 2.
2. Navigate to: **samples/gps**
3. Open the MicroPython file: *gpsdemo1.py*
4. Using the editor of your choice, edit the MicroPython file. At the top of the file, enter the user name and password for your Remote Manager developer account. The correct location is indicated in the comments in the file.

Step 4: Run the program

1. Rename the file you edited in Step 3 from *gpsdemo1.py* to *main.py*.
2. Copy the renamed file onto your device's root filesystem directory.
3. Copy the following three modules from the locations specified below into your device's **/lib** directory:
   - From the **/lib** directory of the Digi xbee-micropython repository: *urequest.py* and *remotemanager.py*
   - From the **/lib/sensor** directory of the Digi xbee-micropython repository: *hdc1080.py*

   Note These modules are required to be able to run the *gpsdemo1.py*.
4. Open XCTU and use the MicroPython Terminal to run the demo.
5. Type <CTRL>-R from the MicroPython prompt to run the code.

Get started with MicroPython

This user guide provides an overview of how to use MicroPython with the XBee 3 802.15.4 RF Module. For in-depth information and more complex code examples, refer to the *Digi MicroPython Programming Guide*. Continue with this user guide for simple examples to get started using MicroPython on the XBee 3 802.15.4 RF Module.

About MicroPython

MicroPython is an open-source programming language based on Python 3.0, with much of the same syntax and functionality, but modified to fit on small devices with limited hardware resources, such as an XBee 3 802.15.4 RF Module.

For more information about MicroPython, see www.micropython.org.

For more information about Python, see www.python.org.

MicroPython on the XBee 3 802.15.4 RF Module

The XBee 3 802.15.4 RF Module has MicroPython running on the device itself. You can access a MicroPython prompt from the XBee 3 802.15.4 RF Module when you install it in an appropriate development board (XBDB or XBIB), and connect it to a computer via a USB cable.

Note MicroPython is only available through the UART interface and does not work with SPI.

Note MicroPython programming on the device requires firmware version 2003 or newer.

The examples in this user guide assume:

- You have XCTU on your computer. See Configure the device using XCTU.
- You have a serial terminal program installed on your computer. For more information, see Use the MicroPython Terminal in XCTU. This requires XCTU 6.3.10 or higher.
- You have an XBee 3 802.15.4 RF Module installed on an appropriate development board such as an XBIB-U-DEV or an XBDB-U-ZB.
- The XBee 3 802.15.4 RF Module is connected to the computer via a USB cable and XCTU recognizes it.

Use XCTU to enter the MicroPython environment

To use the XBee 3 802.15.4 RF Module in the MicroPython environment:

1. Use XCTU to add the device(s); see Configure the device using XCTU and Add devices to XCTU.
2. The XBee 3 802.15.4 RF Module appears as a box in the **Radio Modules** information panel. Each module displays identifying information about itself.
3. Click this box to select the device and load its current settings.

   Note To ensure that MicroPython is responsive to input, Digi recommends setting the XBee UART baud rate to 115200 baud. To set the UART baud rate, select **115200 [7]** in the **BD** field and click the **Write** button. We strongly recommend using hardware flow control to avoid data loss, especially when pasting large amounts of code or text. For more information, see UART flow control.

4. To put the XBee 3 802.15.4 RF Module into MicroPython mode, in the **AP** field select **MicroPython REPL [4]** and click the **Write** button  .
5. Note which COM port the XBee 3 802.15.4 RF Module is using, because you will need this information when you use the MicroPython terminal.

Use the MicroPython Terminal in XCTU

You can use the MicroPython Terminal to communicate with the XBee 3 802.15.4 RF Module when it is in MicroPython mode.[1] This requires XCTU 6.3.10 or higher. To enter MicroPython mode, follow the steps in Use XCTU to enter the MicroPython environment. To use the MicroPython Terminal:

1. Click the **Tools** drop-down menu and select **MicroPython Terminal**. The terminal window opens.
2. Click **Open** to open the Serial Port Configuration window.
3. In the **Select the Serial/USB port** area, click the COM port that the device uses.
4. Verify that the baud rate and other settings are correct.
5. Click **OK**. The **Open** icon changes to **Close**, indicating that the device is properly connected.

If the **>>>** prompt appears, you are connected properly. You can now type or paste MicroPython code in the terminal.

MicroPython examples

This section provides examples of how to use some of the basic functionality of MicroPython with the XBee 3 802.15.4 RF Module.

Example: hello world

1. At the MicroPython **>>>** prompt, type the Python command: **print("Hello, World!")**
2. Press **Enter** to execute the command. The terminal echos back **Hello, World!**

Example: enter MicroPython paste mode

In the following examples it is helpful to know that MicroPython supports paste mode, where you can copy a large block of code from this user guide and paste it instead of typing it character by character. To use paste mode:

1. Copy the code you want to run. For example, copy the following code that is the code from the "Hello world" example:

```
print("Hello World")
```

Note You can easily copy and paste code from the online version of this guide. Use caution with the PDF version, as it may not maintain essential indentations.

2. In the terminal, at the MicroPython **>>>** prompt type **Ctrl**-+**E** to enter paste mode. The terminal displays **paste mode; Ctrl-C to cancel, Ctrl-D to finish**.
3. Right-click in the MicroPython terminal window and click **Paste** or press **Ctrl+Shift+V** to paste.
4. The code appears in the terminal occupying one line. Each line starts with its line number and three "**=**" symbols. For example, line 1 starts with **1===**.

[1] See Other terminal programs if you do not use the MicroPython Terminal in XCTU.

5. If the code is correct, press **Ctrl**+**D** to run the code; “Hello World” should print.

Note If you want to exit paste mode without running the code, or if the code did not copy correctly, press **Ctrl**+**C** to cancel and return to the normal MicroPython **>>>** prompt).

Example: use the time module

The time module is used for time-sensitive operations such as introducing a delay in your routine or a timer.

The following time functions are supported by the XBee 3 802.15.4 RF Module:

- **ticks_ms()** returns the current millisecond counter value. This counter rolls over at 0x40000000.
- **ticks_diff()** compares the difference between two timestamps in milliseconds.
- **sleep()** delays operation for a set number of seconds.
- **sleep_ms()** delays operation for a set number of milliseconds.
- **sleep_us()** delays operation for a set number of microseconds.

Note The standard **time.time()** function cannot be used, because this function produces the number of seconds since the epoch. The XBee3 module lacks a realtime clock and cannot provide any date or time data.

The following example exercises the various sleep functions and uses **ticks_diff()** to measure duration:

```
import time

start = time.ticks_ms()  # Get the value from the millisecond counter

time.sleep(1)            # sleep for 1 second
time.sleep_ms(500)       # sleep for 500 milliseconds
time.sleep_us(1000)      # sleep for 1000 microseconds

delta = time.ticks_diff(time.ticks_ms(), start)

print("Operation took {} ms to execute".format(delta))
```

Example: AT commands using MicroPython

AT commands control the XBee 3 802.15.4 RF Module. The "AT" is an abbreviation for "attention", and the prefix "AT" notifies the module about the start of a command line. For a list of AT commands that can be used on the XBee 3 802.15.4 RF Module, see AT commands.

MicroPython provides an **atcmd()** method to process AT commands, similar to how you can use Command mode or API frames.

The **atcmd()** method accepts two parameters:

1. The two character AT command, entered as a string.
2. An optional second parameter used to set the AT command value. If this parameter is not provided, the AT command is queried instead of being set. This value is an integer, bytes object, or string, depending on the AT command.

Note The **xbee.atcmd()** method does not support the following AT commands: **IS**, **AS**, **ED**, **ND**, or **DN**.

The following is example code that queries and sets a variety of AT commands using **xbee.atcmd()**:

```
import xbee

# Set the NI string of the radio
xbee.atcmd("NI", "XBee3 module")

# Configure a destination address using two different data types
xbee.atcmd("DH", 0x0013A200)           # Hex
xbee.atcmd("DL", b'\x12\x25\x89\xF5')  # Bytes

# Read some AT commands and display the value and data type:
print("\nAT command parameter values:")
commands =["DH", "DL", "NI", "CK"]
for cmd in commands:
       val = xbee.atcmd(cmd)
       print("{}: {:20} of type {}".format(cmd, repr(val), type(val)))
```

This example code outputs the following:

```
AT command parameter values:
DH: b'\x00\x13\xa2\x00'  of type <class 'bytes'>
DL: b'\x12%\x89\xf5'     of type <class 'bytes'>
NI: 'XBee3 module'       of type <class 'str'>
CK: 65535                of type <class 'int'>
```

Note Parameters that store values larger than 16-bits in length are represented as bytes. Python attempts to print out ASCII characters whenever possible, which can result in some unexpected output (such as the "%" in the above output). If you want the output from MicroPython to match XCTU, you can use the following example to convert bytes to hex:

```
dl_value = xbee.atcmd("DL")
hex_dl_value = hex(int.from_bytes(dl_value, 'big'))
```

MicroPython networking and communication examples

This section provides networking and communication examples for using MicroPython with the XBee 3 802.15.4 RF Module.

802.15.4 networks with MicroPython

For small networks, it is suitable to use MicroPython on every node. However, there are some inherit limitations that may prevent you from using MicroPython on some heavily trafficked nodes:

- When running MicroPython, any received messages will be stored in a small receive queue. This queue only has room for 4 packets and must be regularly read to prevent data loss. For networks that will be generating a lot of traffic, the data aggregator may need to operate in API mode in order to capture all incoming data.

For the examples in this section, the devices should be pre-configured with identical network settings so that RF communication is possible. To follow the upcoming examples, we need to configure a second XBee 3 802.15.4 RF Module to use MicroPython.

XCTU only allows a single MicroPython terminal. We will be running example code on both modules, which requires a second terminal window.

Open a second instance of XCTU, and configure a different XBee 3 device for MicroPython following the steps in Use XCTU to enter the MicroPython environment.

Example: network Discovery using MicroPython

The **xbee.discover()** method returns an iterator that blocks while waiting for results, similar to executing an **ND** request. For more information, see ND (Network Discover).

Each result is a dictionary with fields based on an **ND** response:

- **sender_nwk**: 16-bit network address.
- **sender_eui64**: 8-byte bytes object with EUI-64 address.
- **parent_nwk**: Set to 0xFFFE on the coordinator and routers; otherwise, this is set to the network address of the end device's parent.
- **node_id**: The device's **NI** value (a string of up to 20 characters, also referred to as Node Identification).
- **node_type**: Value of 0, 1 or 2 for coordinator, router, or end device.
- **device_type**: The device's 32-bit **DD** value, also referred to as Digi Device Type; this is used to identify different types of devices or hardware.
- **rssi**: Relative signal strength indicator (in dBm) of the node discovery request packet received by the sending node.

Note When printing the dictionary, fields for **device_type**, **sender_nwk** and **parent_nwk** appear in decimal form. You can use the MicroPython **hex()** method to print an integer in hexadecimal. Check the function code for **format_eui64** from the Example: communication between two XBee 3 802.15.4 modules topic for code to convert the **sender_eui64** field into a hexadecimal string with a colon between each byte value.

Use the following example code to perform a network discovery:

```
import xbee, time

# Set the network discovery options to include self
xbee.atcmd("NO", 2)
xbee.atcmd("AC")
time.sleep(.5)

# Perform Network Discovery and print out the results
print ("Network Discovery in process...")
nodes = list(xbee.discover())
if nodes:
       for node in nodes:
              print("\nRadio discovered:")
              for key, value in node.items():
                     print("\t{:<12} : {}".format(key, value))

# Set NO back to the default value
xbee.atcmd("NO", 0)
xbee.atcmd("AC")
```

This produces the following output from two discovered nodes:

```
Radio discovered:
        rssi          : -63
        node_id       : Coordinator
```

```
        device_type    : 1179648
        parent_nwk     : 65534
        sender_nwk     : 0
        sender_eui64   : b'\x00\x13\xa2\xff h\x98T'
        node_type      : 0

Radio discovered:
        rssi           : -75
        node_id        : Router
        device_type    : 1179648
        parent_nwk     : 65534
        sender_nwk     : 23125
        sender_eui64   : b'\x00\x13\xa2\xffh\x98c&'
        node_type      : 1
```

Examples: transmitting data

This section provides examples for transmitting data using MicroPython. These examples assume you have followed the above examples and the two radios are on the same network.

Example: transmit message

Use the **xbee** module to transmit a message from the XBee 3 Zigbee device. The **transmit()** function call consists of the following parameters:

1. The Destination Address, which can be any of the following:
   - Integer for 16-bit addressing
   - 8-byte bytes object for 64-bit addressing
   - Constant **xbee.ADDR_BROADCAST** to indicate a broadcast destination
   - Constant **xbee.ADDR_COORDINATOR** to indicate the coordinator
2. The Message as a character string.

If the message is sent successfully, **transmit()** returns **None**. If the transmission fails due to an ACK failure or lack of free buffer space on the receiver, the sent packet will be silently discarded.

Example: transmit a message to the network coordinator

1. From the router, access the MicroPython environment.
2. At the MicroPython >>> prompt, type **import xbee** and press **Enter**.
3. At the MicroPython >>> prompt, type **xbee.transmit(xbee.ADDR_COORDINATOR, "Hello World!")** and press **Enter**.
4. On the coordinator, you can issue an **xbee.receive()** call to output the received packet.

Example: transmit custom messages to all nodes in a network

This program performs a network discovery and sends the message **'Hello <Destination Node Identifier>!'** to individual nodes in the network. For more information, see Example: network Discovery using MicroPython.

```
import xbee

# Perform a network discovery to gather destination address:
print("Discovering remote nodes, please wait...")
node_list = list(xbee.discover())
if not node_list:
```

```
        raise Exception("Network discovery did not find any remote devices")

for node in node_list:
        dest_addr = node['sender_nwk']  # 'sender_eui64' can also be used
        dest_node_id = node['node_id']
        payload_data = "Hello, " + dest_node_id + "!"

        try:
                print("Sending \"{}\" to {}".format(payload_data, hex(dest_addr)))
                xbee.transmit(dest_addr, payload_data)
        except Exception as err:
                print(err)

print("complete")
```

Receiving data

Use the **receive()** function from the **xbee** module to receive messages. When MicroPython is active on a device (**AP** is set to 4), all incoming messages are saved to a receive queue within MicroPython. This receive queue is limited in size and only has room for 4 messages. To ensure that data is not lost, it is important to continuously iterate through the receive queue and process any of the packets within.

If the receive queue is full and another message is sent to the device, it will not acknowledge the packet and the sender generates a failure status of 0x24 (Address not found).

The **receive()** function returns one of the following:

- None: No message (the receive queue is empty).
- Message dictionary consisting of:
  - **sender_eui64**: 64-bit address (as a "bytes object") of the sending node.
  - **source_ep**: source endpoint as an integer.
  - **dest_ep**: destination endpoint as an integer.
  - **cluster**: cluster id as an integer.
  - **profile**: profile id as an integer.
  - **broadcast**: True or False depending on whether the frame was broadcast or unicast.
  - **payload**: "Bytes object" of the payload. This is a bytes object instead of a string, because the payload can contain binary data.

Example: continuously receive data

In this example, the **format_packet()** helper formats the contents of the dictionary and **format_eui64()** formats the bytes object holding the EUI-64. The **while** loop shows how to poll for packets continually to ensure that the receive buffer does not become full.

```
def format_eui64(addr):
    return ':'.join('%02x' % b for b in addr)

def format_packet(p):
    type = 'Broadcast' if p['broadcast'] else 'Unicast'
    print("%s message from EUI-64 %s (network 0x%04X)" % (type,
        format_eui64(p['sender_eui64']), p['sender_nwk']))
    print("    from EP 0x%02X to EP 0x%02X, Cluster 0x%04X, Profile 0x%04X:" %
        (p['source_ep'], p['dest_ep'], p['cluster'], p['profile']))
    print(p['payload'])
```

```
import xbee, time
while True:
        print("Receiving data...")
        print("Press CTRL+C to cancel.")
        p = xbee.receive()
        if p:
                format_packet(p)
        else:
                time.sleep(0.25)   # wait 0.25 seconds before checking again
```

If this node had previously received a packet, it outputs as follows:

```
Unicast message from EUI-64 00:13:a2:00:41:74:ca:70 (network 0x6D81)
    from EP 0xE8 to EP 0xE8, Cluster 0x0011, Profile 0xC105:
b'Hello World!'
```

Note Digi recommends calling the **receive()** function in a loop so no data is lost. On modules where there is a high volume of network traffic, there could be data lost if the messages are not pulled from the queue fast enough.

Example: communication between two XBee 3 802.15.4 modules

This example combines all of the previous examples and represents a full application that configures a network, discovers remote nodes, and sends and receives messages.

First, we will upload some utility functions into the flash space of MicroPython so that the following examples will be easier to read.

Complete the following steps to compile and execute utility functions using flash mode on both devices:

1. Access the MicroPython environment.
2. Press **Ctrl + F**.
3. Copy the following code:

```
import xbee, time
# Utility functions to perform XBee 3 802.15.4 operations
def format_eui64(addr):
       return ':'.join('%02x' % b for b in addr)

def format_packet(p):
       type = 'Broadcast' if p['broadcast'] else 'Unicast'
       print("%s message from EUI-64 %s (network 0x%04X)" %
               (type, format_eui64(p['sender_eui64']), p['sender_nwk']))
       print("from EP 0x%02X to EP 0x%02X, Cluster 0x%04X, Profile 0x%04X:" %
               (p['source_ep'], p['dest_ep'], p['cluster'], p['profile']))
       print(p['payload'],"\n")

def network_status():
       # If the value of AI is non zero, the module is not connected to a network
       return xbee.atcmd("AI")
```

4. At the MicroPython 1^^^ prompt, right-click and select the **Paste** option.
5. Press **Ctrl+D** to finish. The code is uploaded to the flash memory and then compiled. At the "Automatically run this code at startup" [Y/N]?" prompt, select **Y**.

6. Press **Ctrl+R** to run the compiled code; this provides access to these utility functions for the next examples.

WARNING! MicroPython code stored in flash is saved in the file system as **main.py**. If the file system has not been formatted, then the following error is generated:
OSError: [Errno 7019] ENODEV
The file system can be formatted in one of three ways:
In XCTU by using the File System Manager.
In Command mode using the **ATFS FORMAT confirm** command—see FS (File System).
In MicroPython by issuing the following code:

```
import os
   os.format()
```

Example code on the coordinator module

The following example code forms an 802.15.4 network as a coordinator, performs a network discovery to find the remote node, and continuously prints out any incoming data.

1. Access the MicroPython environment.
2. Copy the following sample code:

```
print("Forming a new 802.15.4 network as a coordinator...")
xbee.atcmd("NI", "Coordinator")
network_settings = {"CE": 1, "A2": 4, "CH": 0x13, "MY": 0xFFFF, "ID": 0x3332,
"EE": 0}
for command, value in network_settings.items():
       xbee.atcmd(command, value)
xbee.atcmd("AC")  # Apply changes
time.sleep(1)

while network_status() != 0:
       time.sleep(0.1)
print("Network Established\n")

print("Waiting for a remote node to join...")
node_list = []
while len(node_list) == 0:
       # Perform a network discovery until the remote joins
       node_list = list(xbee.discover())
print("Remote node found, transmitting data")

for node in node_list:
       dest_addr = node['sender_eui64'] # using 64-bit addressing
       dest_node_id = node['node_id']
       payload_data = "Hello, " + dest_node_id + "!"

       print("Sending \"{}\" to {}".format(payload_data, hex(dest_addr)))
       xbee.transmit(dest_addr, payload_data)

# Start the receive loop
print("Receiving data...")
print("Hit CTRL+C to cancel")
while True:
       p = xbee.receive()
       if p:
```

```
                format_packet(p)
        else:
                time.sleep(0.25)
```

3. Press **Ctrl+E** to enter paste mode.
4. At the **MicroPython >>>** prompt, right-click and select the **Paste** option. Once you paste the code, it executes immediately.

Example code on the remote module

The following example code joins the 802.15.4 network from the previous example, and continuously prints out any incoming data. This device also sends its temperature data every 5 seconds to the coordinator address.

1. Access the MicroPython environment.
2. Copy the following sample code:

```
print("Joining network as an end device...")
xbee.atcmd("NI", "End Device")
network_settings = {"CE": 0, "A1": 4, "CH": 0x13, "ID": 0x3332, "EE": 0}
for command, value in network_settings.items():
        xbee.atcmd(command, value)
xbee.atcmd("AC")  # Apply changes
time.sleep(1)

while network_status() != 0:
        time.sleep(0.1)
print("Connected to Network\n")

last_sent = time.ticks_ms()
interval = 5000  # How often to send a message

# Start the transmit/receive loop
print("Sending temp data every {} seconds".format(interval/1000))
while True:
        p = xbee.receive()
        if p:
                format_packet(p)
        else:
                # Transmit temperature if ready
                if time.ticks_diff(time.ticks_ms(), last_sent) > interval:
                                temp = "Temperature: {}C".format(xbee.atcmd("TP"))
                                print("\tsending " + temp)
                                try:
                                        xbee.transmit(xbee.ADDR_COORDINATOR, temp)
                                except Exception as err:
                                        print(err)
                                last_sent = time.ticks_ms()
                time.sleep(0.25)
```

3. Press **Ctrl+E** to enter paste mode.
4. At the **MicroPython >>>** prompt, right-click and select the **Paste** option. Once you paste the code, it executes immediately.

Exit MicroPython mode

To exit MicroPython mode:

1. In the XCTU MicroPython terminal, click the green **Close** button .
2. Click **Close** at the bottom of the terminal to exit the terminal.
3. In XCTU's Configuration working mode , change **AP API Enable** to another mode and click the **Write** button . We recommend changing to Transparent mode **[0]**, as most of the examples use this mode.

Other terminal programs

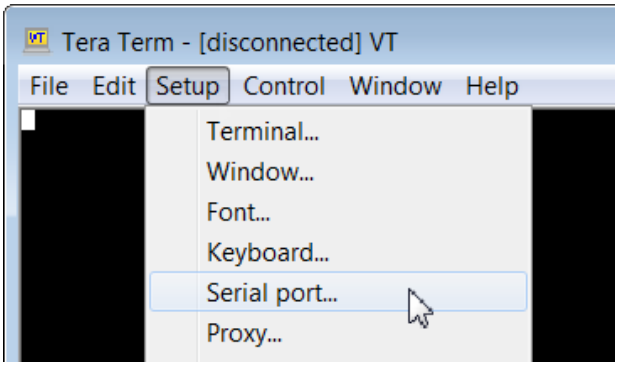
If you do not use the MicroPython terminal in XCTU, you can use other terminal programs to communicate with the XBee 3 802.15.4 RF Module. If you use Microsoft Windows, follow the instructions for Tera Term; if you use Linux, follow the instructions for picocom. To download these programs:

- Tera Term for Windows, see ttssh2.osdn.jp/index.html.en.
- Picocom for Linux, see developer.ridgerun.com/wiki/index.php/Setting_up_Picocom_-_Ubuntu
- Source code and in-depth information, see github.com/npat-efault/picocom.

Tera Term for Windows

With the XBee 3 802.15.4 RF Module in MicroPython mode (**AP** = **4**), you can access the MicroPython prompt using a terminal.

1. Open Tera Term. The **Tera Term: New connection** window appears.
2. Click the **Serial** radio button to select a serial connection.
3. From the **Port:** drop-down menu, select the COM port that the XBee 3 802.15.4 RF Module is connected to.
4. Click **OK**. The **COMxx - Tera Term VT** terminal window appears and Tera Term attempts to connect to the device at a baud rate of 9600 bps. The terminal will not allow communication with the device since the baud rate setting is incorrect. You must change this rate as it was previously set to 115200 bps.
5. Click **Setup** and **Serial Port**. The **Tera Term: Serial port setup** window appears.



6. In the **Tera Term: Serial port setup** window, set the parameters to the following values:
   - **Port**: Shows the port that the XBee 3 802.15.4 RF Module is connected on.
   - **Baud rate**: 115200

- **Data**: 8 bit
- **Parity**: none
- **Stop**: 1 bit
- **Flow control**: hardware
- **Transmit delay**: N/A

7. Click **OK** to apply the changes to the serial port settings. The settings should go into effect right away.
8. To verify that local echo is not enabled and that extra line-feeds are not enabled:
   a. In Tera Term, click **Setup** and select **Terminal**.
   b. In the **New-line** area of the **Tera Term: Serial port setup** window, click the **Receive** drop-down menu and select **AUTO** if it does not already show that value.
   c. Make sure the **Local echo** box is not checked.
9. Click **OK**.
10. Press **Ctrl**+**B** to get the MicroPython version banner and prompt.

```
MicroPython v1.9.3-716-g507d0512 on 2018-02-20; XBee3 802.15.4 with EFR32MG
Type "help()" for more information.
>>>
```

Now you can type MicroPython commands at the **>>>** prompt.

Use picocom in Linux

With the XBee 3 802.15.4 RF Module in MicroPython mode (**AP** = **4**), you can access the MicroPython prompt using a terminal.

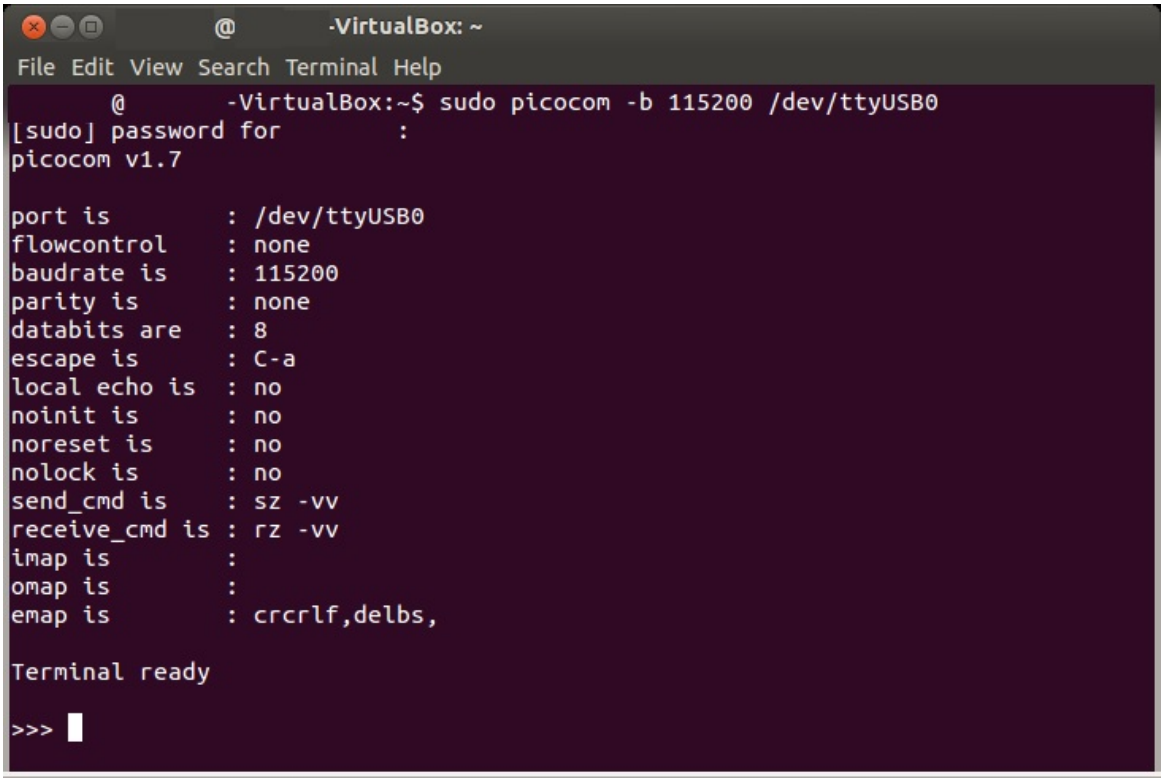
Note The user must have read and write permission for the serial port the XBee 3 802.15.4 RF Module is connected to in order to communicate with the device.

1. Open a terminal in Linux and type **picocom -b 115200 /dev/ttyUSB0**. This assumes you have no other USB-to-serial devices attached to the system.
2. Press **Ctrl**+**B** to get the MicroPython version banner and prompt. You can also press **Enter** to bring up the prompt.

If you do have other USB-to-serial devices attached:

1. Before attaching the XBee 3 802.15.4 RF Module, check the directory **/dev/** for any devices named **ttyUSBx**, where **x** is a number. An easy way to list these is to type: **ls /dev/ttyUSB***. This produces a list of any device with a name that starts with **ttyUSB**.
2. Take note of the devices present with that name, and then connect the XBee 3 802.15.4 RF Module.
3. Check the directory again and you should see one additional device, which is the XBee 3 802.15.4 RF Module.
4. In this case, replace **/dev/ttyUSB0** at the top with **/dev/ttyUSB<number>**, where **<number>** is the new number that appeared.

   It connects and shows "Terminal ready".



You can now type MicroPython commands at the **>>>** prompt.

Micropython help ()

When you type the **help()** command at the prompt, it provides a link to online help, control commands and also usage examples.

```
>>> help()
Welcome to MicroPython!
For online docs please visit http://docs.micropython.org/.
Control commands:
CTRL-A        -- on a blank line, enter raw REPL mode
CTRL-B        -- on a blank line, enter normal REPL mode
CTRL-C        -- interrupt a running program
CTRL-D        -- on a blank line, reset the REPL
CTRL-E        -- on a blank line, enter paste mode
CTRL-F        -- on a blank line, enter flash upload mode
For further help on a specific object, type help(obj)
For a list of available modules, type help('modules')
------------------------------------------------------
------------------------------------------------------
```

When you type **help('modules')** at the prompt, it displays all available Micropython modules.

```
------------------------------------------------------
>>> help('modules')
__main__            io                  time            uos
array               json                ubinascii       ustruct
binascii            machine             uerrno          utime
```

```
builtins            micropython         uhashlib            xbee
errno               os                  uio
gc                  struct              ujson
hashlib             sys                 umachine

Plus any modules on the filesystem

-------------------------------------------------------
```

When you import a module and type **help()** with the module as the object, you can query all the functions that the object supports.

```
-------------------------------------------------------
>>> import sys
>>> help(sys)
object <module 'sys'> is of type module
  __name__ -- sys
  path -- ['', '/flash', '/flash/lib']
  argv -- ['']
  version -- 3.4.0
  version_info -- (3, 4, 0)
  implementation -- ('micropython', (1, 10, 0))
  platform -- xbee3-802.15.4
  byteorder -- little
  maxsize -- 2147483647
  exit -- <function>
  stdin -- <io.FileIO 0>
  stdout -- <io.FileIO 1>
  stderr -- <io.FileIO 2>
  modules -- {}
  print_exception -- <function>
-------------------------------------------------------
```

Secure access

By default, the XBee 3 802.15.4 RF Module is easy to configure and allows for rapid prototyping. For deployment, you can encrypt networks to prevent unauthorized access. This can prevent entities outside of the network from accessing data on that network. Some customers may also desire a way to restrict communication between nodes from inside the same network.

There are three ways to secure your device against unauthorized access:

- Secure remote session
- Disable functionality

Secure session protects against external man-in-the middle attacks by requiring remote devices to authenticate before they are allowed to make configuration changes.

You can also disable device functionality in order to prevent unexpected malicious use of the product. for example disable MicroPython so that remote code cannot be uploaded and executed.

Secure Sessions

Secure Sessions provide a way to password-protect communication between two nodes on a network above and beyond the security of the network itself. With secure sessions, a device can 'log in', or create a session with another device that is encrypted and only readable by the two nodes involved. By restricting certain actions—such as remote AT commands or FOTA updates—to only be allowed over one of these secure sessions, you can make it so access to the network does not allow network configuration. A password must be set and the proper bits of SA (Secure Access) must be set to enable this feature.

The following definitions relate to secure Sessions:

Term	Definition
Client	The device that is attempting to log in and send secured data or commands is called the client.
Server	The device that is being logged into and will receive secured data or commands is called the server.
Secure Session	A secure connection between a server and a client where the pair can send and receive encrypted data that only they can decrypt.
Secure Remote Password (SRP)	Name of the authentication protocol used to create the secure connection between the nodes.
Salt	A random value generated as part of the authentication process.
Verifier	A value derived from a given salt and password.

Configure the secure session password for a device

For a device to act as a secure session server it needs to have a password configured. The password is configured on the server in the form of a salt and verifier used for the SRP authentication process. The salt and verifier can be configured in XCTU by selecting the **Secure Session Authentication** option.

We recommend using XCTU to set a password which will then generate the salt and verifier parameters, although the salt and verifier values can also be set manually. See *S (Secure Session Salt) and *V, *W, *X, *Y (Secure Session Verifier) for more information.

Note There is not an enforced password length. We recommend a minimum length of at least eight characters. The password should not exceed 64 characters, as it will exceed the maximum length of an API frame.

Start a secure session

A secure session can only be started in API mode. Once you have been authenticated you may send data in API mode or Transparent mode, but API mode is the recommended way to communicate.

To start a secure session:

1. Send a type Secure Session Control - 0x2E to your local client device with the address of the server device (not a broadcast address), the options bit field set to **0x00**, the timeout for the session, and the password that was previously set on the server.
2. The client and server devices will send/exchange several packets to authenticate the session.

3. When authentication is complete, the client device will output a Secure Session Response - 0xAE to indicate whether the login was a success or failure.

At this point if authentication was successful, the secure session is established and the client can send secured data to the server until the session times out.

Note A device can have one outgoing session—a session in which the node is a client—at a time. Attempting to start a new session while a session is already in progress automatically ends the previous session.

Note A device can have up to four incoming sessions—sessions in which the device is a server—at a time. Once that number has been reached, additional authentication requests are rejected until one of the active sessions ends.

End a secure session

A client can end a session by either waiting for the timeout to expire or by ending it manually. To end a session, send a Secure Session Control - 0x2E to the local client device with bit 0 of the options field set and with no password.

The device ends the outgoing secure session with the node whose address is specified in the type 0x2E frame. This frame can be sent even if the node does not have a session with the specified address—the device will send a message to the specified server prompting it to clear out any incoming session data related to the client (this can be used if the server and client fall out of sync. For example, if the client device unexpectedly loses power during a session.

Sending a type 0x2E frame with the logout option bit set, and the address field set to the broadcast address will end whatever outgoing session is currently active on the client and broadcast a request to all servers to clear any incoming session data related to that client.

Secured remote AT commands

Secure a node against unauthorized remote configuration

Secured Access is enabled by setting bits of SA (Secure Access). Additionally, an SRP Salt (***S**) and verifier (***V**, ***W**, ***X**, ***Y**) must be set. You can use XCTU to generate the salt and verifier based on a password.

Configure a node with a salt and verifier

In this example, the password is **pickle**.

1. The salt is randomly generated and the verifier is derived from the salt and password as follows:

   ***S = 0x1938438E**

   ***V = 0x0771F57C397AE4019347D36FD1B9D91FA05B2E5D7365A161318E46F72942A45D**

   ***W = 0xD4E44C664B5609C6D2BE3258211A7A20374FA65FC7C82895C6FD0B3399E73770**

   ***X = 0x63018D3FEA59439A9EFAE3CD658873F475EAC94ADF7DC6C2C005b930042A0B74**

   ***Y = 0xAEE84E7A00B74DD2E19E257192EDE6B1D4ED993947DF2996CAE0D644C28E8307**

Note The salt and verifier will not always be the same even if the same password is used to generate them.

2. Enforce secure access for Remote AT Commands by setting Bit 1 of the **SA** command:

   SA = **0x02**

3. Write the configuration to flash using WR (Write).

WARNING! Make sure that this step is completed. If your device resets for any reason and *S and SA are not written to flash they will revert to defaults, rendering the node open to insecure access.

4. From now on, any attempt to issue a Remote AT Command Request - 0x17 to this device will be rejected with a **0x0B** status unless a secure session is established first.

Remotely configure a node that has been secured

In the example above a node is secured against unauthorized remote configuration. In this instance, the secured node acts as a Secure Session Server (remote). The sequence below describes how a Secure Session Client (local) can authenticate and securely configure the server remotely.

Establish a secure session using the password that was set on the server node

1. Generate a Secure Session Control - 0x2E.

- The destination address must match the 64-bit address (**SH** + **SL**) of the remote server.
- Since you are logging in, leave the options field as **0x00**.
- Set a five minute timeout, which should give sufficient time for ad hoc configuration. The units are in tenths of a second, so **0x0BB8** gives you five minutes.
- The options are set for a fixed duration, so after the five minutes expire, both the server and client emit a modem status indicating the session ended.
- Enter the original password used to generate the verifier from the random salt above.

2. Pass the type 0x2E Control frame into the serial interface of the local client:

- For example, to log into a Secure Session server at address **0013A200 417B2162** for a five minute duration using the password **pickle**, use the following frame:
  7E 00 12 2E 00 13 A2 00 41 7B 21 62 00 0B B8 70 69 63 6B 6C 65 A2

3. Wait for a Secure Session Response - 0xAE to indicate the session establishment succeeded or failed with the reason.

- The address of the remote that is responding and the status is included in the response.
- For example, the response to the request above is as follows:
  7E 00 0B AE 00 00 13 A2 00 41 7B 21 62 **00** 5D. The 0**x00** status indicates success.

4. Send remote AT Commands to the remote server using the Remote AT Command Request - 0x17 with bit 4 of the Command Options field set. Bit 4 indicates the AT command should be sent securely.

Note If you are using 802.15.4 firmware you must send secured packets using the device's 64 bit address. To do so, set MY (16-bit Source Address) to **0xFFFF**.

Send data to a secured remote node

The process to send secured data is very similar to remotely configuring a node. The following steps show how a client node can authenticate with a server node and send data securely.

1. Send a Secure Session Control - 0x2E to the client node with:
   - The server's 64-bit address.
   - The desired timeout.
   - The options field set to **0x00** for fixed timeout login or to **0x04** for inter-packet timeout refresh login.
   - The password of the server node.
2. Wait for the Secure Session Response - 0xAE to determine if the the authentication was successful.
3. Data can now be sent securely with Transmit Request - 0x10 and Explicit Addressing Command Request - 0x11 provided that:
   - Bit 4 in the transmit options field is set to indicate that the data should be sent encrypted.
4. The returned Receive Packet - 0x90 and Explicit Receive Indicator - 0x91 receive options fields should also have bit 4 set.

Note The maximum payload per transmission size is reduced by four bytes due to the additional encryption overhead. NP (Maximum Packet Payload Bytes) will not reflect this change when the session is going on.

Note If you use 802.15.4 firmware you must use 64-bit addressing to send secured packets. To do so, set MY (16-bit Source Address) to **0xFFFF**.

A node can be secured against emitting data out the serial port that was received insecurely via the **SA** command. This means that a remote node will not emit any serial data if it was received insecurely (TO (Transmit Options) bit 4 was not set). This includes any data in Transparent mode, **0x80**, **0x90** and **0x91** frames.

Note When a device rejects a data transmission (0x80, 0x90, 0x91, or Transparent data) because of its **SA** configuration, it does not send an error back to the sender. This means that data transmissions to a device give a success status even if they are rejected.

End a session from a server

If bit 3 of AZ (Extended API Options) is set, the server emits an extended modem status (whenever a client establishes a session with it) that includes the 64-bit address of the client. Using these statuses the MCU connected to the server can keep track of sessions established with the server. To end a session from the server do the following:

1. Send a Secure Session Control - 0x2E to the server node with:
   - The client's 64-bit address.
   - The options field set to **0x02** for server side session termination.
   - Set the timeout to **0x0000**.
2. Wait for the Secure Session Response - 0xAE to determine if the termination was successful.
   - The client will emit a modem status **0x3C** (Session Ended).
   - The server will also emit a modem status (or an extended modem status depending on **AZ**) of **0x3C** (Session Ended).

Note The 64-bit address can be set to the broadcast address to end all incoming sessions.

Note This functionality can be used to end orphaned client-side sessions—in case the server unexpectedly reset for some reason.

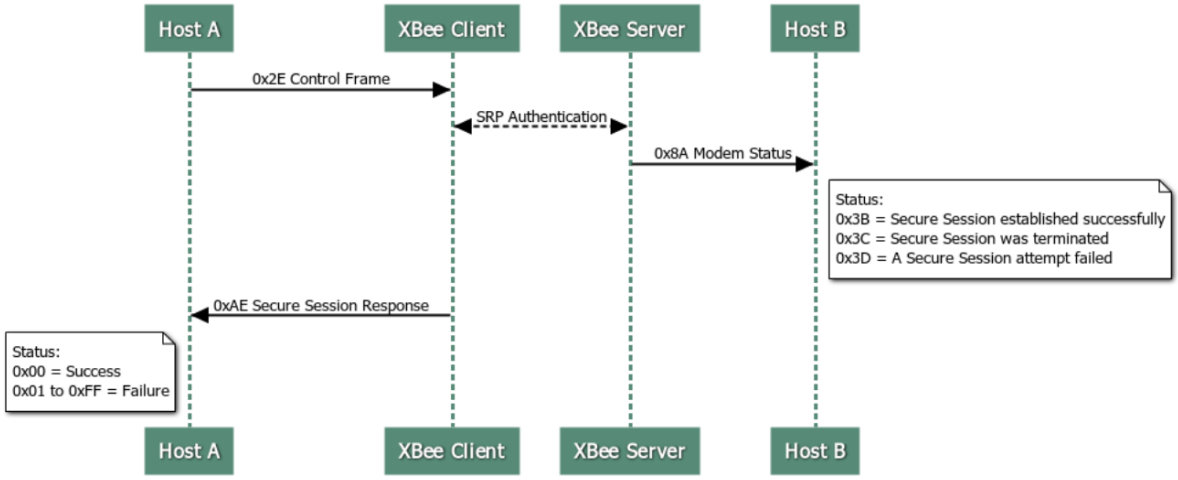
Secure Session API frames

Secure Session can only be established from a node that is operating in API mode (MicroPython support is forthcoming). The server-side can be in Transparent mode, but the client must be in API mode. Once a session has been established between a client and server node, the client can be transitioned to Transparent mode; and if bit 4 of **TO** is set, the client will encrypt data sent in Transparent mode for the duration of session.

There are four frames that are used for controlling and observing a secure session.

- Secure Session Control - 0x2E: This frame is passed to the client that wishes to log into or out of a server. Any attempt to use the Control frame will generate a response frame.
- Secure Session Response - 0xAE: This frame returns the status of the previously sent 0x2E frame indicating whether it was successful or not.
- Modem Status - 0x8A: The server will also emit a modem status whenever an attempt succeeds, fails, or was terminated. The client will also emit modem statuses if the session times out.
- Extended Modem Status - 0x98: If bit 3 of **AZ** is set then modem statuses will be replaced with extended modem statuses. These frames will contain the status that caused them to be emitted as well as the address of the node that initiated the session, the session options, and the timeout value.

Frame exchanges:



Secure transmission failures

This section describes the error messages you can see when trying to send a secure packet.

Data Frames - 0x10 and 0x11 frames

- Response frame type: Extended Transmit Status - 0x8B

Possible error statuses:

Status	Description	Reason
0x34	No Secure Session Connection	The sending node does not have an active session with the destination node.
0x35	Encryption Failure	The encryption process failed. Only likely to be seen when using manual SRP and when an invalid encryption parameter was passed in.

Remote AT Commands- 0x17 frames

- Response frame type: Remote AT Command Response- 0x97

Possible error statuses:

Status	Description	Reason
0x0B	No Secure Session Connection	The sending node does not have an active session with the destination node.
0x0C	Encryption Error	There was an internal encryption error on the radio.
0x0D	TO Bit Not Set	The client has a session with the server but forgot to set the **TO** bit.

File system

For detailed information about using MicroPython on the XBee 3 802.15.4 RF Module refer to the *Digi MicroPython Programming Guide*.

Overview of the file system

XBee 3 802.15.4 RF Module firmware versions 2003 and later include support for storing files in internal flash memory.

CAUTION! You need to format the file system if upgrading a device that originally shipped with older firmware. You can use XCTU, AT commands or MicroPython for that initial format or to erase existing content at any time.

Note To use XCTU with file system, you need XCTU 6.4.0 or newer.

See **FS FORMAT confirm** in FS (File System) and ensure that the format is complete.

Directory structure

The XBee 3 802.15.4 RF Module's internal flash appears in the file system as **/flash**, the only entry at the root level of the file system. Files and directories other than **/flash** cannot be created within the root directory, only within **/flash**. By default **/flash** contains a lib directory intended for MicroPython modules.

Paths

The XBee 3 802.15.4 RF Module stores all of its files in the top-level directory **/flash**. On startup, the **ATFS** commands and MicroPython each use that directory as their current working directory. When specifying the path to a file or directory, it is interpreted as follows:

- Paths starting with a forward slash are "absolute" and must start with **/flash** to be valid.
- All other paths are relative to the current working directory.
- The directory **..** refers to the parent directory, so an operation on **../filename.txt** that takes place in the directory **/flash/test** accesses the file **/flash/filename.txt**.
- The directory **.** refers to the current directory, so the command **ATFS ls .** lists files in the current directory.
- Names are case-insensitive, so **FILE.TXT**, **file.txt** and **FiLe.TxT** all refer to the same file.
- File and directory names are limited to 64 characters, and can only contain letters, numbers, periods, dashes and underscores. A period at the end of the name is ignored.
- The full, absolute path to a file or directory is limited to 255 characters.

Limitations

The file system on the XBee 3 802.15.4 RF Module has a few limitations when compared to conventional file systems:

- When a file on the file system is deleted, the space it was using is only reclaimed if it is found at the end of the file system. Deleted data that is contiguous with the last placed deleted file is also reclaimed.
- The file system can only have one file open for writing at a time.
- The file system cannot create new directories while a file is open for writing.

- Files cannot be renamed.
- The contents of the file system will be lost when any firmware update is performed. See OTA file system upgrades for information on how to put files on a device after a FOTA update.

XCTU interface

XCTU releases starting with 6.4.0 include a **File System Manager** in the **Tools** menu. You can upload files to and download files from the device, in addition to renaming and deleting existing files and directories. See the File System manager tool section of the *XCTU User Guide* for details of its functionality.

Get started with BLE

Bluetooth® Low Energy (BLE) is a RF protocol that enables you to connect your XBee device to another device. Both devices must have BLE enabled.

For example, you can use your cellphone to connect to your XBee device, and then from your phone, configure and program the device.

Digi created the Digi XBee Mobile SDK, a set of libraries, examples and documentation that help you develop mobile applications to interact with XBee devices through their BLE interface. For this purpose, we provide two easy-to-use libraries that allow you to create XBee mobile native apps:

- XBee Library for Xamarin, to develop cross-platform mobile applications using C# language (iOS and Android).
- XBee Library for Android, to develop Android applications using Java

The XBee is the server and allows client devices, such as a cellphone, to configure the XBee or data transfer with the User Data Relay frame. The XBee cannot communicate with another XBee over BLE, as the XBee is strictly a BLE server. The possibilities are:

- XBee 3: can communicate with mobile devices over BLE
- XBee 3: can communicate with third party devices such as the Nordic nRF and SiLabs BGM over BLE
- XBee 3: cannot communicate with another XBee 3 over BLE

Enable BLE on the XBee 3 802.15.4 RF Module

To enable BLE on a XBee 3 802.15.4 RF Module and verify the connection:

1. Set up the XBee 3 802.15.4 RF Module and make sure to connect the antenna to the device.
2. Enable BLE and configure the BLE password.
3. Get the Digi XBee Mobile phone application.
4. Connect with BLE and configure your XBee 3 device.

Note The BLE protocol is disabled on the XBee 3 802.15.4 RF Module by default. You can create a custom factory default configuration that ensures BLE is always enabled. See Custom configuration: Create a new factory default.

Enable BLE and configure the BLE password

Some of the latest XBee 3 devices support Bluetooth Low Energy (BLE) as an extra interface for configuration. If you want to use this feature, you have to enable BLE. You must also enable security by setting a password on the XBee 3 802.15.4 RF Module in order to connect, configure, or send data over BLE.

Use XCTU to configure the BLE password. Make sure you have installed or updated XCTU to version 6.4.2 or newer. Earlier versions of XCTU do not include the BLE configuration features. See Download and install XCTU for installation instructions.

Before you begin, you should determine the password you want to use for BLE on the XBee 3 802.15.4 RF Module and store it in a secure place. We recommend a secure password of at least eight characters and a random combination of letters, numbers, and special characters. We recommend using a security management tool such as LastPass or Keepass for generating and storing passwords for many devices.

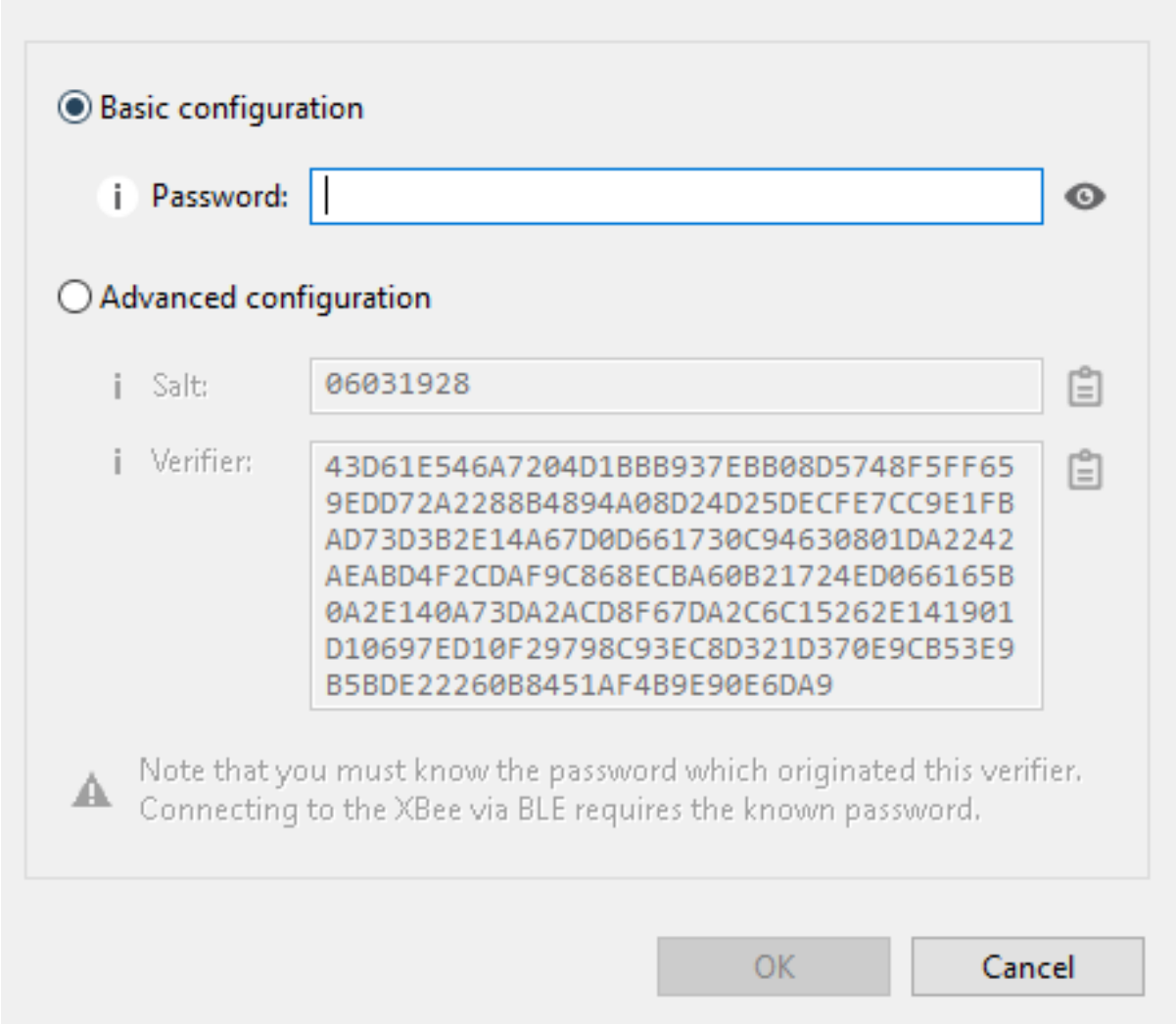
Note When you enter the BLE password in XCTU, the salt and verifier values are calculated as you set your password. For more information on how these values are used in the authentication process, see BLE Unlock Request - 0x2C.

1. Launch XCTU .
2. Switch to Configuration working mode .
3. Select a BLE compatible radio module from the device list.
4. Select **Enabled[1]** from the **BT Bluetooth Enable** command drop-down.

5. Click the **Write setting** button . The **Bluetooth authentication not set** dialog appears.

Note If BLE has been previously configured, the **Bluetooth authentication not set** dialog does not appear. If this happens, click **Configure** in the Bluetooth Options section to display the **Configure Bluetooth Authentication** dialog.

6. Click **Configure** in the dialog. The **Configure Bluetooth Authentication** dialog appears.
7. In the **Password** field, type the password for the device. As you type, the **Salt** and **Verifier** fields are automatically calculated and populated in the dialog as shown above. This password is used when you connect to this XBee device via BLE using the Digi XBee Mobile app.



8. Click **OK** to save the configuration.

Get the Digi XBee Mobile phone application

To see the nearby devices that have BLE enabled, you must get the free Digi XBee Mobile application from the iOS App Store or Google Play and downloaded to your phone.

1. On your phone, go to the App store.
2. Search for: **Digi XBee Mobile**.
3. Download and install the app.

The Digi is compatible with the following operating systems and versions:

- Android 5.0 or higher
- iOS 11 or higher

Connect with BLE and configure your XBee 3 device

You can use the Digi XBee Mobile application to verify that BLE is enabled on your XBee device.

1. Get the Digi XBee Mobile phone application.
2. Open the Digi XBee Mobile application. The **Find XBee devices** screen appears and the app automatically begins scanning for devices. All nearby devices with BLE enabled are displayed in a list.
3. Scroll through the list to find your XBee device.
   The first time you open the app on a phone and scan for devices, the device list contains only the name of the device and the BLE signal strength. No identifying information for the device displays. After you have authenticated the device, the device information is cached on the phone. The next time the app on this phone connects to the XBee device, the IMEI for the device displays in the app device list.

Note The IMEI is derived from the **SH** and **SL** values.

4. Tap the XBee device name in the list. A password dialog appears.
5. Enter the password you previously configured for the device in XCTU.
6. Tap **OK**. The **Device Information** screen displays. You can now scroll through the settings for the device and change the device's configuration as needed.

BLE reference

BLE advertising behavior and services

When the Bluetooth radio is enabled, periodic BLE advertisements are transmitted. The advertisement data includes the product name in the Complete Local Name field. When an XBee device connects to the Bluetooth radio, the BLE services are listed:

- Device Information Service
- XBee API BLE Service

Device Information Service

The standard Device Information Service is used. The Manufacturer, Model, and Firmware Revision characters are provided inside the service.

XBee API BLE Service

You can configure the XBee 3 802.15.4 RF Module through the BLE interface using API frame requests and responses. The API frame format through Bluetooth is equivalent to setting **AP** = **1** and transmitting the frames over the UART or SPI interface. API frames can be executed over Bluetooth regardless of the AP setting.

The BLE interface allows these frames:

- BLE Unlock Request - 0x2C
- User Data Relay Input - 0x2D
- BLE Unlock Response - 0xAC
- Local AT Command Request - 0x08
- Queue Local AT Command Request - 0x09

This API reference assumes that you are familiar with Bluetooth and GATT services. The specifications for Bluetooth are an open standard and can be found at the following links:

- Bluetooth Core Specifications: bluetooth.com/specifications/bluetooth-core-specification
- Bluetooth GATT: bluetooth.com/specifications/gatt/generic-attributes-overview

The XBee API BLE Service contains two characteristics: the API Request characteristic and the API Response characteristic. The UUIDs for the service and its characteristics are listed in the table below.

Characteristic	UUID
API Service UUID	53da53b9-0447-425a-b9ea-9837505eb59a
API Request Characteristic UUID	7dddca00-3e05-4651-9254-44074792c590
API Response Characteristic UUID	f9279ee9-2cd0-410c-81cc-adf11e4e5aea

API Request characteristic

UUID: 7dddca00-3e05-4651-9254-44074792c590

Permissions: Writeable

XBee API frames are broken into chunks and transmitted sequentially to the request characteristic using write operations. Valid frames are then processed and the result is returned through indications on the response characteristic.

API frames do not need to be written completely in a single write operation to the request characteristic. In fact, Bluetooth limits the size of a written value to 3 bytes smaller than the configured Maximum Transmission Unit (MTU), which defaults to 23, meaning that by default, you can only write 20 bytes at a time.

After connecting you must send a valid Bluetooth Unlock API Frame in order to authenticate the connection. If the BLE Unlock API - 0x2C frame has not been executed, all other API frames are silently ignored and are not processed.

API Response characteristic

UUID: f9279ee9-2cd0-410c-81cc-adf11e4e5aea

Permissions: Readable, Indicate

Responses to API requests made to the request characteristic are returned through the response characteristics. This characteristic cannot be read directly.

Response data is presented through indications on this characteristic. Indications are acknowledged and re-transmitted at the BLE link layer and application layer and provide a robust transport for this data.

Configure the XBee 3 802.15.4 RF Module

Software libraries

One way to communicate with the XBee 3 802.15.4 RF Module is by using a software library. The libraries available for use with the XBee 3 802.15.4 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.

Firmware over-the-air (FOTA) update

The XBee 3 802.15.4 RF Module supports FOTA updates using XCTU version 6.3.0 or higher. For instructions on performing a FOTA firmware update with XCTU, see How to update the firmware of your modules in the XCTU User Guide.

FOTA capability is only available when **MM** (Mac Mode) = **0** or **3**.

Custom defaults

Custom defaults allow you to preserve a subset of the device configuration parameters even after returning to default settings using RE (Restore Defaults). This can be useful for settings that identify the device—such as NI (Node Identifier)—or settings that could make remotely recovering the device difficult if they were reset—such as ID (Extended PAN ID).

Note You must send these commands as local AT commands, they cannot be set using Remote AT Command Request - 0x17.

Set custom defaults

Use %F (Set Custom Default) to set custom defaults. When the XBee 3 802.15.4 RF Module receives **%F** it takes the next command it receives and applies it to both the current configuration and the custom defaults.

To set custom defaults for multiple commands, send a **%F** before each command.

Restore factory defaults

!C (Clear Custom Defaults) clears all custom defaults, so that RE (Restore Defaults) will restore the device to factory defaults. Alternatively, R1 (Restore Factory Defaults) restores all parameters to factory defaults without erasing their custom default values.

Limitations

There is a limitation on the number of custom defaults that can be set on a device. The number of defaults that can be set depends on the size of the saved parameters and the devices' firmware version. When there is no more room for custom defaults to be saved, any command sent immediately after a **%F** returns an error.

Custom configuration: Create a new factory default

You can create a custom configuration that is used as a new factory default. This feature is useful if, for example, you need to maintain certain settings for manufacturing or want to ensure a feature is always enabled. When you use RE (Restore Defaults) to perform a factory reset on the device, the custom configuration is set on the device after applying the original factory default settings.

For example, by default Bluetooth is disabled on devices. You can create a custom configuration in which Bluetooth is enabled by default. When you use **RE** to reset the device to the factory defaults, the Bluetooth configuration set to the custom configuration (enabled) rather than the original factory default (disabled).

The custom configuration is stored in non-volatile memory. You can continue to create and save custom configurations until the XBee 3 802.15.4 RF Module's memory runs out of space. If there is no space left to save a configuration, the device returns an error.

You can use !C (Clear Custom Defaults) to clear or overwrite a custom configuration at any time.

Set a custom configuration

1. Open XCTU and load your device.
2. Enter Command mode.
3. Perform the following process for each configuration that you want to set as a factory default.
   a. Send the Set Custom Default command, **AT%F**. This command enables you to enter a custom configuration.
   b. Send the custom configuration command. For example: **ATBT 1**. This command sets the default for Bluetooth to enabled.

Clear all custom configuration on a device

After you have set configurations using %F (Set Custom Default), you can return all configurations to the original factory defaults.

1. Open XCTU and load the device.
2. Enter Command mode.
3. Send **AT!C**.

XBee bootloader

You can update firmware on the XBee 3 802.15.4 RF Module serially. This is done by invoking the XBee 3 bootloader and transferring the firmware image using XMODEM.

This process is also used for updating a local device's firmware using XCTU.

XBee devices use a modified version of Silicon Labs' Gecko bootloader. This bootloader version supports a custom entry mechanism that uses module pins DIN, DTR/SLEEP_RQ, and RTS.

To invoke the bootloader using hardware flow control lines, do the following:

1. Set DTR/SLEEP_RQ low (CMOS0V) and RTS high.
2. Send a serial break to the DIN pin and power cycle or reset the module.
3. When the device powers up, set DTR/SLEEP_RQ and DIN to low (CMOS0V) and RTS should be high.
4. Terminate the serial break and send a carriage return at 115200 baud to the device.

5. If successful, the device sends the Silicon Labs' Gecko bootloader menu out the DOUT pin at 115200 baud.
6. You can send commands to the bootloader at 115200 baud.

Note Disable hardware flow control when entering and communicating with the bootloader.

All serial communications with the module use 8 data bits, no parity bit, and 1 stop bit.

You can also invoke the bootloader from the XBee application by sending %P (Invoke Bootloader).

Send a firmware image

After invoking the bootloader, a menu is sent out the UART at 115200 baud. To upload a firmware image through the UART interface:

1. Look for the bootloader prompt **BL** > to ensure the bootloader is active.
2. Send an ASCII **1** character to initiate a firmware update.
3. After sending a **1**, the device waits for an XModem CRC upload of a .gbl image over the serial line at 115200 baud. Send the .gbl file to the device using standard XMODEM-CRC.

If the firmware image is successfully loaded, the bootloader outputs a “complete” string. Invoke the newly loaded firmware by sending a **2** to the device.

If the firmware image is not successfully loaded, the bootloader outputs an "aborted string". It return to the main bootloader menu. Some causes for failure are:

- Over 1 minute passes after the command to send the firmware image and the first block of the image has not yet been sent.
- A power cycle or reset event occurs during the firmware load.
- A file error or a flash error occurs during the firmware load. The following table contains errors that could occur during the XMODEM transfer.

Error	Cause	Workaround
0x18	This error is observed when a serial upload attempt has been abruptly discontinued by invoking **Ctrl**+**C** and subsequently another attempt is made to upload a gbl by pressing **1** on the bootloader menu.	Press **2** on the bootloader menu. The bootloader performs a reboot and the menu gets displayed again. Now press **1** and begin uploading the gbl.

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.

XBee Multi Programmer

The XBee Multi Programmer is a combination of hardware and software that enables partners and distributors to program multiple Digi Radio frequency (RF) devices simultaneously. It provides a fast and easy way to prepare devices for distribution or large networks deployment.

The XBee Multi Programmer board is an enclosed hardware component that allows you to program up to six RF modules thanks to its six external XBee sockets. The XBee Multi Programmer application communicates with the boards and allows you to set up and execute programming sessions. Some of the features include:

- Each XBee Multi Programmer board allows you to program up to six devices simultaneously. Connect more boards to increase the programming concurrency.
- Different board variants cover all the XBee form factors to program almost any Digi RF device.

Download the XBee Multi Programmer application from: digi.com/support/productdetail?pid=5641

See the *XBee Multi Programmer User Guide* for more information.

Modes

Transparent operating mode

Devices operate in this mode by default. The device acts as a serial line replacement when it is in Transparent operating mode. The device queues all 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 Command mode.

Transparent operating mode is not available when using the SPI interface; see SPI operation.

Serial-to-RF packetization

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

1. No serial characters are received for the amount of time determined by the **RO** (Packetization Timeout) parameter. If **RO** = **0**, packetization begins when a character is received.
2. The maximum number of characters that will fit in an RF packet is received. There are a number of factors that determine payload size. You can query the NP (Maximum Packet Payload Bytes) to determine the maximum payload size based on current configuration. For more information, see Maximum payload.
3. The Command mode Sequence—**GT** + **CC** + **GT**—is received; this is any data in the serial receive buffer received before the sequence is transmitted. For more information, see Enter Command mode.

If the device cannot immediately transmit (for instance, if it is already receiving RF data), the serial data is stored in the serial receive buffer. The data is packetized and sent at any **RO** timeout or when **NP** bytes are received.

If the serial receive buffer becomes full, hardware flow control must be implemented in order to prevent overflow—loss of data between the host and device.

API operating mode

Application programming interface (API) operating mode is an alternative to Transparent mode. 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. 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 control over the operation of the radio network or when you need to know which node a data packet is from. The device communicates UART or SPI data in packets, also known as API frames. This mode allows for structured communications with serial devices.

For more information, see API mode overview.

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 3 802.15.4 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 3 802.15.4 RF Module are controlled by the AP (API Enable) 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 unless using SPI for the serial interface.

Enter Command mode

When using the default configuration values for **GT** and **CC**, you must enter **+++** preceded and followed by one second of silence—no input—to enter Command mode. However, both **GT** and **CC** are configurable. This means that the silence before and after the escape sequence—**GT**—and the escape characters themselves—**CC**—can be changed. For example, if **GT** is **5DC** and **CC** is **31**, then Command mode can be entered by typing **111** preceded and followed by 1.5 seconds of silence. 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 3 802.15.4 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 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 (UART 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.

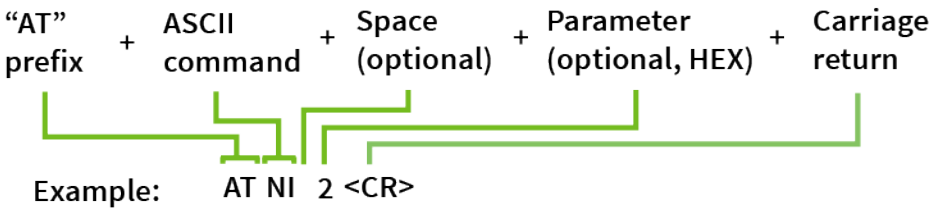
Note You must assert $\overline{\text{RTS}}$ for both of these methods, otherwise the device enters the bootloader.

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 **2**.

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 3 802.15.4 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). In this case, changes are only applied following a reset. The **WR** command by itself does not apply changes.
   or:
3. Exit Command mode. You can exit Command mode in two ways: Either enter the **CN** command or wait for Command mode to timeout as specified by the **CT** parameter.

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 an RE (Restore Defaults) followed by **WR** to restore parameters back to their factory defaults. The next time the device is reset the default settings are applied.

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.

Idle mode

When not receiving or transmitting data, the XBee 3 802.15.4 RF Module is in Idle mode. During Idle mode, the device listens for valid data on both the RF and serial ports.

If configured for Sleep support, the XBee 3 802.15.4 RF Module only transitions to a low power state when in Idle mode.

Transmit mode

Transmit mode is the mode in which the device is transmitting data. This typically happens after data is received from the serial port.

Receive mode

This is the default mode for the XBee 3 802.15.4 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.

Serial communication

Serial interface

The XBee 3 802.15.4 RF Module interfaces to a host device through a serial port. The device can communicate through its serial port:

- Through logic and voltage compatible universal asynchronous receiver/transmitter (UART).
- Through a level translator to any serial device, for example through an RS-232 or USB interface board.
- Through SPI, as described in SPI communications.

Serial receive buffer

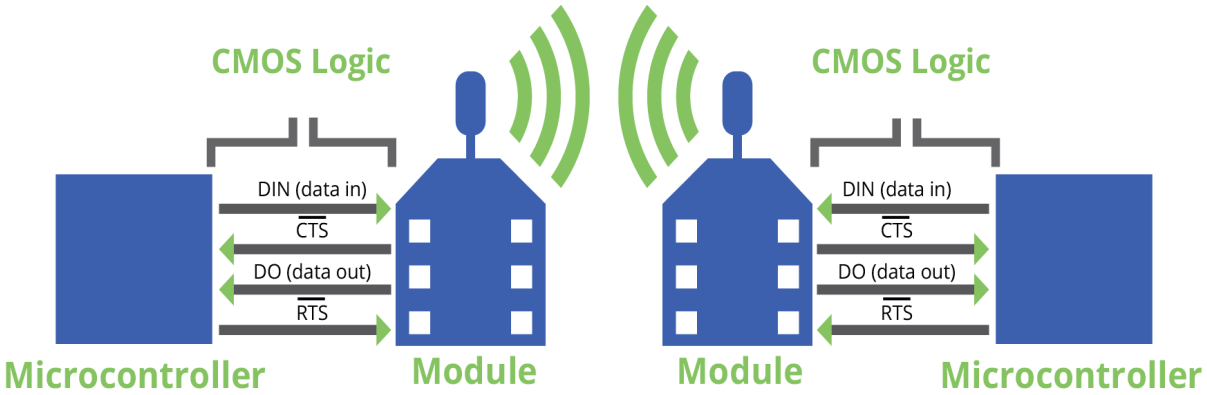
When serial data enters the device through the DIN pin or the SPI_MOSI pin, it stores the data in the serial receive buffer until the device can process it. Under certain conditions, the device may not be able to process data in the serial receive buffer immediately. If large amounts of serial data are sent to the device such that the serial receive buffer overflows, then the device discards all incoming data until it is able to process the data in the buffer. The size of the Serial receive buffer is 960 Bytes; the serial buffer may be reduced in size if RAM requirements cannot be met in future firmware releases. If the UART is in use, you can avoid this by the host side by honoring clear-to-send (CTS) flow control.

Serial transmit buffer

When the device receives RF data, it moves the data into the serial transmit buffer and sends it out the UART. If the serial transmit buffer becomes full and the system buffers are also full, then it drops the entire RF data packet. The size of the Serial transmit buffer is 304 Bytes; the serial buffer may be reduced in size if RAM requirements cannot be met in future firmware releases. 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 data flow

Devices that have a UART interface connect directly to the pins of the XBee 3 802.15.4 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.



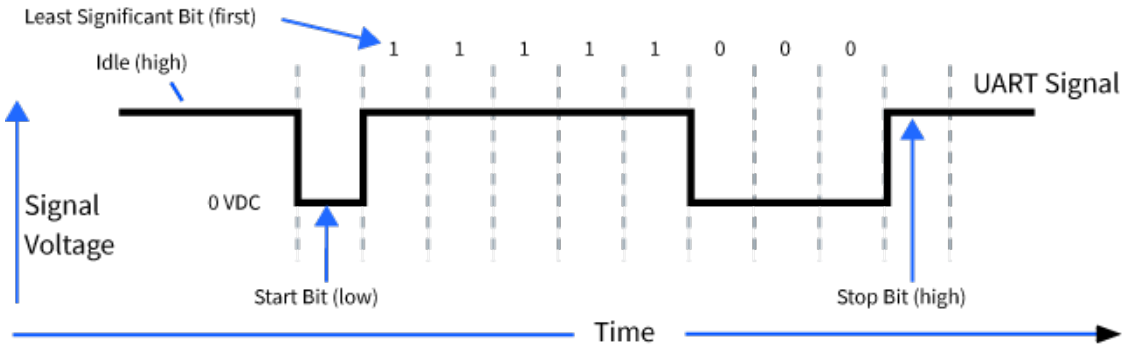
For more information about hardware specifications for the UART, see the *XBee 3 Hardware Reference Manual*.

Serial data

A device sends data to the XBee 3 802.15.4 RF Module's UART 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 3 802.15.4 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.

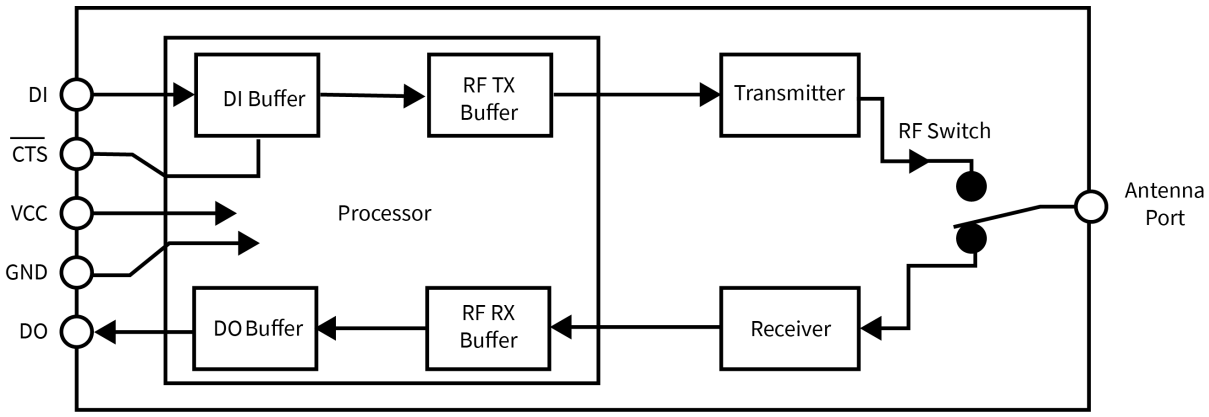


You can configure the UART baud rate, parity, and stop bits settings on the device with the **BD**, **NB**, and **SB** commands respectively. For more information, see UART interface commands.

Flow control

The XBee 3 802.15.4 RF Module maintains buffers to collect serial and RF data that it receives. The serial receive buffer collects incoming serial characters and holds them until the device can process them. The serial transmit buffer collects the data it receives via the RF link until it transmits that data out the serial port. The following figure shows the process of device buffers collecting received serial data.

Use D6 (DIO6/RTS Configuration) and D7 (DIO7/CTS Configuration) to set flow control.



Clear-to-send ($\overline{\text{CTS}}$) flow control

If you enable $\overline{\text{CTS}}$ flow control (D7 (DIO7/$\overline{\text{CTS}}$ Configuration)), when the serial receive buffer is more than **FT** bytes full, the device de-asserts $\overline{\text{CTS}}$ (sets it high) to signal to the host device to stop sending

serial data. The device reasserts CTS after the serial receive buffer has less than **FT** bytes in it. See FT (Flow Control Threshold) to configure and read this threshold.

RTS flow control

If you set D6 (DIO6/RTS Configuration) 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.

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.

Cases in which the DO buffer may become full, resulting in dropped RF packets:

1. If the RF data rate is set higher than the interface data rate of the device, the device may receive data faster than it can send the data to the host. Even occasional transmissions from a large number of devices can quickly accumulate and overflow the transmit buffer.
2. If the host does not allow the device to transmit data out from the serial transmit buffer due to being held off by hardware flow control.

SPI operation

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

SPI communications

The XBee 3 802.15.4 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.

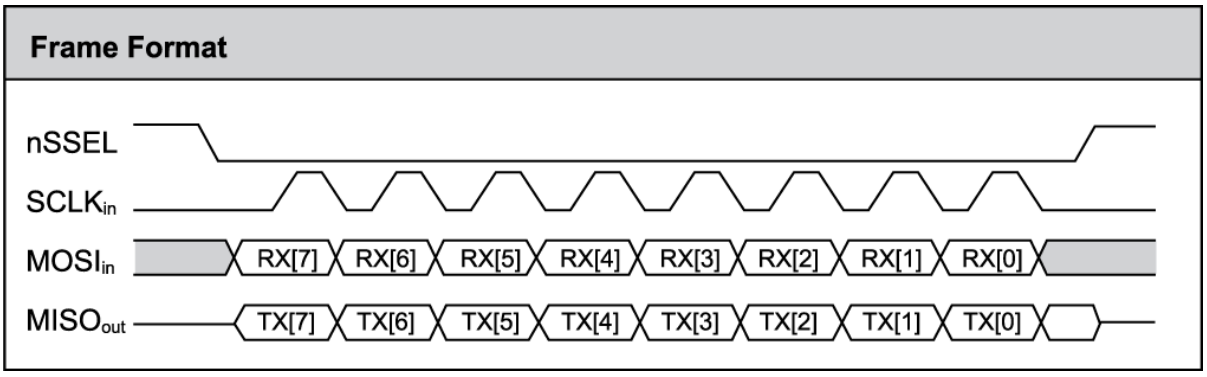
Refer to the XBee 3 Hardware Reference Guide for the pinout of your device.

Signal	Direction	Function
SPI_MOSI (Master Out, Slave In)	Input	Inputs serial data from the master
SPI_MISO (Master In, Slave Out)	Output	Outputs serial data to the master
SPI_SCLK (Serial Clock)	Input	Clocks data transfers on MOSI and MISO
SPI_$\overline{\text{SSEL}}$ (Slave Select)	Input	Enables serial communication with the slave
SPI_$\overline{\text{ATTN}}$ (Attention)	Output	Alerts the master that slave has data queued to send. The XBee 3 802.15.4 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 5 MHz (burst) are possible.
- Data is most significant bit (MSB) first; bit 7 is the first bit of a byte sent over the interface.
- 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.



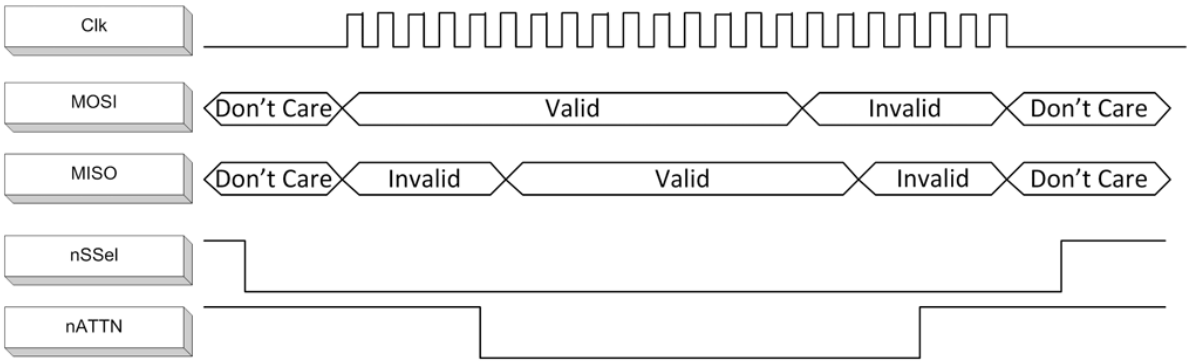
SPI mode is chip to chip communication. We do not supply a SPI communication interface on the XBee development evaluation boards included in the development kit.

Full duplex operation

When using SPI on the XBee 3 802.15.4 RF Module the device uses API operation without escaped characters to packetize data. The device ignores the configuration of **AP** because SPI does not operate in any other mode. SPI is a full duplex protocol, even when data is only available in one direction. This means that whenever a device receives data, it also transmits, and that data is normally invalid. Likewise, whenever a device transmits data, invalid data is probably received. To determine whether or not received data is invalid, the firmware places the data in 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 sends data to the slave and the slave has valid data to send in the middle of receiving data from the master, a full duplex operation occurs, 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.

The following figure 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 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. Without control of SPI_SSEL while using it for sleep request, the device may go to sleep at inopportune times.

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.

Select the SPI port

To force SPI mode on through-hole devices, hold DOUT/DIO13 low while resetting the device until SPI_ATTN asserts. This causes the device to disable the UART and go straight into SPI communication mode. Once configuration is complete, the device queues a modem status frame to the SPI port, which causes the SPI_ATTN line to assert. The host can use this to determine that the SPI port is configured properly.

On surface-mount devices, forcing DOUT low at the time of reset has no effect. To use SPI mode on the SMT modules, assert the SPI_SSEL low after reset and before any UART data is input.

Forcing DOUT low on TH devices forces the device to enable SPI support by setting the following configuration values:

Through-hole	Micro and Surface-mount	SPI signal
D1 (DIO1/ADC1/TH_SPI_ATTN Configuration)	P9 (DIO19/SPI_ATTN Configuration)	$\overline{\text{ATTN}}$
D2 (DIO2/ADC2/TH_SPI_CLK Configuration)	P8 (DIO18/SPI_CLK Configuration)	SCLK
D3 (DIO3/ADC3/TH_SPI_SSEL Configuration)	P7 (DIO17/SPI_SSEL Configuration)	$\overline{\text{SSEL}}$
D4 (DIO4/TH_SPI_MOSI Configuration)	P6 (DIO16/SPI_MOSI Configuration)	MOSI
P2 (DIO12/TH_SPI_MISO Configuration)	P5 (DIO15/SPI_MISO Configuration)	MISO

Note The $\overline{\text{ATTN}}$ signal is optional—you can still use SPI mode if you disable the SPI_$\overline{\text{ATTN}}$ pin (**D1** on through-hole or **P9** on surface-mount devices).

As long as the host does not issue a **WR** command, these configuration values revert to previous values after a power-on reset. If the host issues a **WR** command while in SPI mode, these same parameters are written to flash, and after a reset the device continues to operate in SPI mode.

If the UART is disabled and the SPI is enabled in the written configuration, then the device comes up in SPI mode without forcing it by holding DOUT low. If both the UART and the SPI are configured (P3 (DIO13/UART_DOUT Configuration) through P9 (DIO19/SPI_ATTN Configuration) are set to **1**) at the time of reset, then output goes to the UART until the host sends the first input to the SPI interface. As soon as the first input comes on the SPI port, then all subsequent output goes to the SPI port and the UART is disabled.

Once you select a serial port (UART or SPI), all subsequent output goes to that port, even if you apply a new configuration. Once the SPI interface is made active, the only way to switch the selected serial port back to UART is to reset the device.

When the master asserts the slave select (SPI_$\overline{\text{SSEL}}$) signal, SPI transmit data is driven to the output pin SPI_MISO, and SPI data is received from the input pin SPI_MOSI. The SPI_$\overline{\text{SSEL}}$ pin has to be asserted to enable the transmit serializer to drive data to the output signal SPI_MISO. A rising edge on SPI_$\overline{\text{SSEL}}$ causes the SPI_MISO line to be tri-stated such that another slave device can drive it, if so desired.

If the output buffer is empty, the SPI serializer transmits the last valid bit repeatedly, which may be either high or low. Otherwise, the device formats all output in API mode 1 format, as described in Operate in API mode. The attached host is expected to ignore all data that is not part of a formatted API frame.

Force UART operation

If you configure a device with only the SPI enabled and no SPI master is available to access the SPI slave port, you can recover the device to UART operation by holding DIN / CONFIG low at reset time. 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 then send the appropriate commands to the device to configure it for UART operation. If you write those parameters, the device comes up with the UART enabled on the next reset.

I/O support

The following topics describe analog and digital I/O line support, line passing and output control.

Legacy support

By default, the XBee 3 802.15.4 RF Module is configured to operate in a legacy configuration. This provides network and application compatibility with XBee S1 802.15.4 and XBee S2C 802.15.4 devices. AO (API Output Options) is used to determine:

- Which Digital I/O lines are sampled
- What sample frame type is used for outgoing transmissions

AO has no affect on received I/O sample data, but will determine the frame type emitted for received serial data.

Previous 802.15.4 firmwares on the XBee S1 and XBee S2C hardware had a limited set of I/O lines available. Valid DIO lines on these devices are from **D0** through **D8**; I/O samples are transmitted over the air using a standard I/O sample packet using a Legacy data format. These platforms do not have an **AO** command and always output sample data in a legacy format if possible.

For the XBee 3 platform, digital I/O has been enhanced to be in parity with DigiMesh and Zigbee. You can now enable up to fourteen digital inputs for sampling: **D0** through **P4** as long as **AO** is not set to **2**. In order to support these additional I/O lines, an enhanced I/O sample packet is sent over the air, which is not compatible with the S1 or S2C.

By default, the XBee 3 802.15.4 RF Module is configured to operate in a legacy configuration with **AO** set to **2**. This allows you to sample **D0** through **D8**. If you configure **D9** through **P4** as digital I/O, they are not sampled unless you set **AO** to **0** or **1**.

For new designs, we recommend setting **AO** to **0** or **1** (Operate in API mode), which allows you to use additional I/O lines for sampling and easily allows you to switch to Zigbee or DigiMesh, as the API and I/O functionality are identical.

This table illustrates the various configuration combinations that are possible and the expected output:

Source	Source AO value	Destination	Destination AO value	Data format	API frame on receiver
XBee 3	0 or 1	XBee 3	0 or 1	Enhanced	I/O Sample Indicator - 0x92
XBee 3	0 or 1	XBee 3	2	Enhanced	I/O Sample Indicator - 0x92
XBee 3	0 or 1	S1 or S2C	N/A	N/A	Legacy devices are unable to interpret the additional sample data and will discard the received packet.
XBee 3	2	XBee 3	0 or 1	Legacy	64-bit I/O Sample Indicator - 0x82 / 16-bit I/O Sample Indicator - 0x83
XBee 3	2	S1 or S2C	N/A	Legacy	64-bit I/O Sample Indicator - 0x82 / 16-bit I/O Sample Indicator - 0x83
S1 or S2C	N/A	XBee 3	0 or 1	Legacy	64-bit I/O Sample Indicator - 0x82 / 16-bit I/O Sample Indicator - 0x83
S1 or S2C	N/A	XBee 3	2	Legacy	64-bit I/O Sample Indicator - 0x82 / 16-bit I/O Sample Indicator - 0x83

Refer to I/O sample data format for more information on the format of the incoming I/O sample data.

Mixed network considerations

If you use a mixed network of XBee 3 and legacy S1 or S2C devices, you must set **AO** to **2** in order to transmit sample data that is compatible with these devices.

Regardless of the **AO** setting, if an XBee 3 802.15.4 RF Module receives an I/O sample packet from an S1 or S2C device, it always outputs the legacy data format.

Digital I/O support

AO (API Output Options) determines the I/O lines available for sampling. By default, **AO** is configured to be compatible with legacy devices.

- Configure **AO** to **0** or **1** to make digital I/O available on lines DIO0 through DIO14 (**D0** - **D9** and **P0** - **P4**).
- Configure **AO** to **2** to make digital I/O available on lines DIO0 through DIO8 (**D0** - **D8**). This provides compatibility with S1 and S2C devices and is the default configuration.

See Legacy support for more information.

Digital sampling is enabled on these pins if configured as **3**, **4**, or **5** with the following meanings:

- 3 is digital input.
  - Use PR (Pull-up/Down Resistor Enable) to enable internal pull up/down resistors for each digital input. Use PD (Pull Up/Down Direction) to determine the direction of the internal pull up/down resistor. All disabled and digital input pins are pulled up by default.
- 4 is digital output low.
- 5 is digital output high.

Function when AO = 0 or 1	Legacy Function when AO = 2	Micro Pin	SMT Pin	TH Pin	AT Command
DIO0	DIO0	31	33	20	D0 (DIO0/ADC0/Commissioning Configuration)
DIO1	DIO1	30	32	19	D1 (DIO1/ADC1/TH_SPI_ATTN Configuration)
DIO2	DIO2	29	31	18	D2 (DIO2/ADC2/TH_SPI_CLK Configuration)
DIO3	DIO3	28	30	17	D3 (DIO3/ADC3/TH_SPI_SSEL Configuration)
DIO4	DIO4	23	24	11	D4 (DIO4/TH_SPI_MOSI Configuration)
DIO5	DIO5	26	28	15	D5 (DIO5/Associate Configuration)
DIO6	DIO6	27	29	16	D6 (DIO6/RTS Configuration)
DIO7	DIO7	24	25	12	D7 (DIO7/CTS Configuration)

Function when AO = 0 or 1	Legacy Function when AO = 2	Micro Pin	SMT Pin	TH Pin	AT Command
DIO8	DIO8	9	10	9	D8 (DIO8/DTR/SLP_Request Configuration)
DIO9	N/A	25	26	13	D9 (DIO9/ON_SLEEP Configuration)
DIO10	N/A	7	7	6	P0 (DIO10/RSSI/PWM0 Configuration)
DIO11	N/A	8	8	7	P1 (DIO11/PWM1 Configuration)
DIO12	N/A	5	5	4	P2 (DIO12/TH_SPI_MISO Configuration)
DIO13	N/A	3	3	2	P3 (DIO13/UART_DOUT Configuration)
DIO14	N/A	4	4	3	P4 (DIO14/UART_DIN Configuration)

I/O sampling is not available for pins **P5** through **P9**. See the *XBee 3 Hardware Reference Manual* for full pinouts and functionality.

Analog I/O support

Analog input is available on **D0** through **D3**. Configure these pins to **2** (ADC) to enable analog sampling.

PWM output is available on **P0** and **P1**, which can be used for Analog line passing. Use M0 (PWM0 Duty Cycle) and M1 (PWM1 Duty Cycle) to set a fixed PWM level.

Function	Micro Pin	SMT Pin	TH Pin	AT Command
ADC0	31	33	20	D0 (DIO0/ADC0/Commissioning Configuration)
ADC1	30	32	19	D1 (DIO1/ADC1/TH_SPI_ATTN Configuration)
ADC2	29	31	18	D2 (DIO2/ADC2/TH_SPI_CLK Configuration)
ADC3	28	30	17	D3 (DIO3/ADC3/TH_SPI_SSEL Configuration)
PWM0	7	7	6	P0 (DIO10/RSSI/PWM0 Configuration)
PWM1	8	8	7	P1 (DIO11/PWM1 Configuration)

AV (Analog Voltage Reference) specifies the analog reference voltage used for the 10-bit ADCs. Analog sample data is represented as a 2-byte value. For a 10-bit ADC, the acceptable range is from **0x0000** to **0x03FF**. To convert this value to a useful voltage level, apply the following formula:

ADC / 1023 (vREF) = Voltage

Note ADCs sampled through MicroPython will have 12-bit resolution.

Example

An ADC value received is 0x01AE; to convert this into a voltage the hexadecimal value is first converted to decimal (0x01AE = 430). Using the default **AV** reference of 1.25 V, apply the formula as follows:

430 / 1023 (1.25 V) = 525 mV

Monitor I/O lines

You can monitor pins you configure as digital input, digital output, or analog input and generate I/O sample data. If you do not define inputs or outputs, no sample data is generated.

Typically, I/O samples are generated by configuring the device to sample I/O pins periodically (based on a timer) or when a change is detected on one or more digital pins. These samples are always sent over the air to the destination address specified with DH (Destination Address High) and DL (Destination Address Low).

You can also gather sample data using on-demand sampling, which allows you to interrogate the state of the device's I/O pins by issuing an AT command. You can do this on either a local or remote device via an AT command request.

The three methods to generate sample data are:

- Periodic sample (IR (Sample Rate))
  - Periodic sampling based on a timer
  - Samples are taken immediately upon wake (excluding pin sleep)
  - Sample data is sent to **DH**+**DL** destination address
  - Can be used with line passing
  - Requires API mode on receiver
- Change detect (IC (DIO Change Detect))
  - Samples are generated when the state of specified digital input pin(s) change
  - Sample data is sent to **DH**+**DL** destination address
  - Can be used with line passing
  - Requires API mode on receiver
- On-demand sample (IS (I/O Sample))
  - Immediately query the device's I/O lines
  - Can be issued locally in Command Mode
  - Can be issued locally or remotely in API mode

These methods are not mutually exclusive and you can use them in combination with each other.

I/O sample data format

AO determines the format of outgoing sample data.

By default, **AO** is configured to be compatible with legacy devices and generates samples using a legacy data format.

Legacy data format

If sample data is generated from an S1 or S2C 802.15.4 XBee or an XBee 3 802.15.4 that has **AO** set to **2**, the format of the sample data will be represented as a series of bytes in the following format which is compatible with the S1 802.15.4 and S2C 802.15.4 devices:

Bytes	Name	Description
1	Sample sets	Number of sample sets. This is determined by IT (Samples before TX) on the source node.
2	Digital and analog channel mask	Indicates which digital I/O and ADC lines have sampling enabled. Each bit corresponds to one digital I/O or ADC line on the device. 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 = ADC0 bit 10 = ADC1 bit 11 = ADC2 bit 12 = ADC3 bit 13 = Reserved bit 14 = Reserved bit 15 = Reserved Example: a channel mask of 0x063C means ADC0, ADC1, DIO2, DIO3, and DIO5 are configured as digital inputs or outputs.
2	Digital data set	Each bit in the digital data set corresponds to a digital bit in the channel mask and indicates the state of the digital pin, whether high (1) or low (0). If the digital portion of the channel mask is **0**, then these two bytes are omitted as no digital I/O lines are enabled. 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 = N/A bit 10 = N/A bit 11 = N/A bit 12 = N/A bit 13 = N/A bit 14 = N/A bit 15 = N/A
2	Analog data set (multiple)	Each enabled ADC line in the analog portion of the channel mask has a separate 2-byte value based on the number of ADC inputs on the originating device. The data starts with AD0 and continues sequentially for each enabled analog input channel up to AD3. If the analog portion of the channel mask is **0**, then no analog sample bytes are included.

Enhanced data format

If you set **AO** to **0** or **1** on the source node, then the data format is represented as a series of bytes in the following format which matches the DigiMesh and Zigbee firmwares:

Bytes	Name	Description
1	Sample sets	Number of sample sets. There is always one sample set per frame.
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 = 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 Example: a digital channel mask of 0x002F means DIO0, 1, 2, 3 and 5 are configured as digital inputs or outputs.
1	Analog channel mask	Indicates which lines have analog inputs enabled for sampling. Each bit in the analog channel mask corresponds to one analog input channel. If a bit is set, then a corresponding 2-byte analog data set is included. bit 0 = AD0/DIO0 bit 1 = AD1/DIO1 bit 2 = AD2/DIO2 bit 3 = AD3/DIO3
2	Digital data set	Each bit in the digital data set corresponds to a bit in the digital channel mask and indicates the digital state of the pin, whether high (1) or low (0). If the digital channel mask is 0x0000, then these two bytes are omitted as no digital I/O lines are enabled.
2	Analog data set (multiple)	Each enabled ADC line in the analog channel mask will have a separate 2-byte value based on the number of ADC inputs on the originating device. The data starts with AD0 and continues sequentially for each enabled analog input channel up to AD3. If the analog channel mask is 0x00, then no analog sample bytes is included.

API frame support

I/O samples generated using Periodic I/O sampling (**IR**) and Digital I/O change detection (**IC**) are transmitted to the destination address specified by **DH** and **DL**. In order to display the sample data,

the receiver must be operating in API mode (**AP** = **1** or **2**). The sample data is represented as an I/O sample API frame.

There are three types of I/O sample frames that are supported by the XBee 3 802.15.4 RF Module:

- 0x92 - Enhanced I/O sample frame
- 0x82 - Legacy 64-bit I/O sample frame
- 0x83 - Legacy 16-bit I/O sample frame

If **AO** = **0** or **1** on the source node, additional I/O lines can be sampled by the source and a 0x92 frame is generated on the destination. In this configuration, the receiver must be an XBee 3, as the XBee S1 and S2C 802.15.4 devices will be unable to interpret the additional sample data.

See I/O Sample Indicator - 0x92 for more information on the frame's format and an example.

If the source node is an XBee S1 or S2C device or an XBee 3 with **AO** set to **2**, the destination node generates either a 0x82 or 0x83 frame depending on whether the source node is operating in a 16-bit or 64-bit configuration. See Addressing modes for more information.

See Legacy support for more information on what configuration options generate the various I/O frames.

On-demand sampling

You can use IS (I/O Sample) to query the current state of all digital I/O and ADC lines on the device and return the sample data as an AT command response. If no inputs or outputs are defined, the command returns an ERROR.

On-demand sampling can be useful when performing initial deployment, as you can send **IS** locally to verify that the device and connected sensors are correctly configured. The format of the sample data matches what is periodically sent using other sampling methods. You can also send **IS** remotely using a remote AT command. When sent remotely from a gateway or server to each sensor node on the network, on-demand sampling can improve battery life and network performance as the remote node transmits sample data only when requested instead of continuously.

If you send **IS** using Command mode, then the device returns a carriage return delimited list containing the I/O sample data. If **IS** is sent either locally or remotely via an API frame, the I/O sample data is presented as the parameter value in the AT command response frame (Description or Remote AT Command Response- 0x97).

Example: Command mode

An **IS** command sent in Command mode returns the following sample data:

This example uses the enhanced I/O data format, if you use the legacy format (**AO** = **2** or data is received from an S1 or S2C device) then refer to the Legacy data format for information on how this data is structured.

Output	Description
01	One sample set
0C0C	Digital channel mask, indicates which digital lines are sampled (0x0C0C = 0000 **11**00 0000 **11**00b = DIO2, 3, 10, 11)

Output	Description
03	Analog channel mask, indicates which analog lines are sampled (0x03 = 0000 00**11**b = AD0, 1)
0408	Digital sample data that corresponds with the digital channel mask 0x0408 = 0000 **01**00 0000 **1**000b = DIO3 and DIO10 are high, DIO2 and DIO11 are low
03D0	Analog sample data for AD0
0124	Analog sample data for AD1

Example: Local AT command in API mode

The **IS** command sent to a local device in API mode would use a Local AT Command Request - 0x08 or Queue Local AT Command Request - 0x09 frame:

7E 00 04 08 53 49 53 08

The device responds with a Description that contains the sample data:

7E 00 0F 88 53 49 53 00 01 0C 0C 03 04 08 03 D0 01 24 68

This example uses the enhanced I/O data format, if you use the legacy format (**AO** = **2** or data is received from an S1 or S2C device) then see the Legacy data format for information on how this data is structured.

Output	Field	Description
7E	Start Delimiter	Indicates the beginning of an API frame
00 0F	Length	Length of the packet
88	Frame type	AT Command response frame
53	Frame ID	This ID corresponds to the Frame ID of the 0x08 request
49 53	AT Command	Indicates the AT command that this response corresponds to 0x49 0x53 = **IS**
00	Status	Indicates success or failure of the AT command **00** = OK if no I/O lines are enabled, this will return 01 (ERROR)

Output	Field	Description
01	I/O sample data	One sample set
0C 0C		Digital channel mask, indicates which digital lines are sampled (0x0C0C = 0000 **11**00 0000 **11**00b = DIO2, 3, 10, 11)
03		Analog channel mask, indicates which analog lines are sampled (0x03 = 0000 00**11**b = AD0, 1)
04 08		Digital sample data that corresponds with the digital channel mask 0x0408 = 0000 **01**00 0000 **10**00b = DIO3 and DIO10 are high, DIO2 and DIO11 are low
03 D0		Analog sample data for AD0
01 24		Analog sample data for AD1
68	Checksum	Can safely be discarded on received frames

Example: Remote AT command in API mode

The **IS** command sent to a remote device with an address of 0013A200 12345678 uses a Remote AT Command Request - 0x17:

7E 00 0F 17 87 00 13 A2 00 12 34 56 78 FF FE 00 49 53 FF

The sample data from the device is returned in a Remote AT Command Response- 0x97 frame with the sample data as the parameter value:

7E 00 19 97 87 00 13 A2 00 12 34 56 78 00 00 49 53 00 01 0C 0C 03 04 08 03 FF 03 FF 50

This example uses the enhanced I/O data format, if you use the legacy format (**AO** = **2** or data is received from an S1 or S2C device) then see Legacy data format for information on how this data is structured.

Output	Field	Description
7E	Start Delimiter	Indicates the beginning of an API frame
00 19	Length	Length of the packet
97	Frame type	Remote AT Command response frame
87	Frame ID	This ID corresponds to the Frame ID of the 0x17 request
0013A200 12345678	64-bit source	The 64-bit address of the node that responded to the request
0000	16-bit source	The 16-bit address of the node that responded to the request
49 53	AT Command	Indicates the AT command that this response corresponds to 0x49 0x53 = **IS**
00	Status	Indicates success or failure of the AT command **00** = **OK** if no I/O lines are enabled, this will return **01** (ERROR)

Output	Field	Description
01	I/O sample data	One sample set
0C 0C		Digital channel mask, indicates which digital lines are sampled (0x0C0C = 0000 1100 0000 1100b = DIO2, 3, 10, 11)
03		Analog channel mask, indicates which analog lines are sampled (0x03 = 0000 0011b = AD0, 1)
04 08		Digital sample data that corresponds with the digital channel mask 0x0408 = 0000 0100 0000 1000b = DIO3 and DIO10 are high, DIO2 and DIO11 are low
03 D0		Analog sample data for AD0
01 24		Analog sample data for AD1
50	Checksum	Can safely be discarded on received frames

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.

Source

Use IR (Sample Rate) to set the periodic sample rate for enabled I/O lines.

- To disable periodic sampling, set **IR** to **0**.
- For all other **IR** values, the device samples data when **IR** milliseconds elapse and transmits the sampled data to the destination address.

The DH (Destination Address High) and DL (Destination Address Low) commands determine the destination address of the I/O samples. You must configure at least one pin as a digital I/O or ADC input on the sending node to generate sample data.

Destination

If the receiving device is operating in API operating mode the I/O sample data format is emitted out of the serial port. Devices that are in Transparent operating mode discard the I/O data samples they receive unless you enable line passing.

I/O sampling upon wake

By default, a device that is configured for sleep (**SM** > **0**) that has at least one digital I/O or ADC enabled transmits an I/O sample upon wake regardless of how **IR** is configured. Sampling upon wake can be disabled by clearing bit 1 of the **SO**. For more information about setting sleep modes, see Sleep modes and SO (Sleep Options).

Multiple samples per packet

IT (Samples before TX) specifies how many I/O samples can be transmitted in a single OTA packet. Any single-byte value (0 - 0xFF) is accepted for input. However, the value is adjusted downward based on

how many I/O samples can fit into a maximum size packet; see Maximum payload. A query of **IT** after changes are applied tells how many I/O samples will actually be gathered.

Since MM (MAC Mode) must be **0** or **3** to send I/O samples, the maximum payload in the best of conditions (short source address, short destination address, and no encryption) is 114 bytes. Seven of those bytes are used by the command header and the I/O header, leaving 107 bytes for I/O samples. The minimum I/O sample is 2 bytes. Therefore the maximum possible usable value for **IT** is 53 (or **0x35**).

Only legacy I/O frames allow for gathering multiple samples. If you set **AO** to **0** or **1**, then **IT** is not applicable and only one sample can be gathered per frame.

Example: Remote AT command in API mode

A device is configured with the following settings:

- **D0** and **D1** are set to ADC (**2**)
- **D3** is configured as a digital input (**3**)
- **AO** is set to **2**, so legacy frames are generated
- **IT** is configured to **3**, so that three samples are gathered per transmission

On the destination node, the following frame is emitted:

7E 00 1A 83 12 34 26 02 03 06 04 00 04 01 28 03 12 00 00 01 58 02 FE 00 04 01 2A 03 A0 94

Output	Field	Description
7E	Start Delimiter	Indicates the beginning of an API frame
00 1A	Length	Length of the packet
83	Frame type	Legacy 16-bit I/O Sample
12 34	16-bit Source Address	The source address of the device that sent the I/O sample
26	RSSI	The 64-bit address of the node that responded to the request
02		
03	Sample sets	The number of samples that are included in this frame
06 04	Channel mask	Mask which indicates which digital and analog lines are enabled. Even though multiple samples are being gathered, there will only ever be one channel mask per frame. (0x0604 = 0000 0110 0000 0100b = ADC0, ADC1, DIO3)

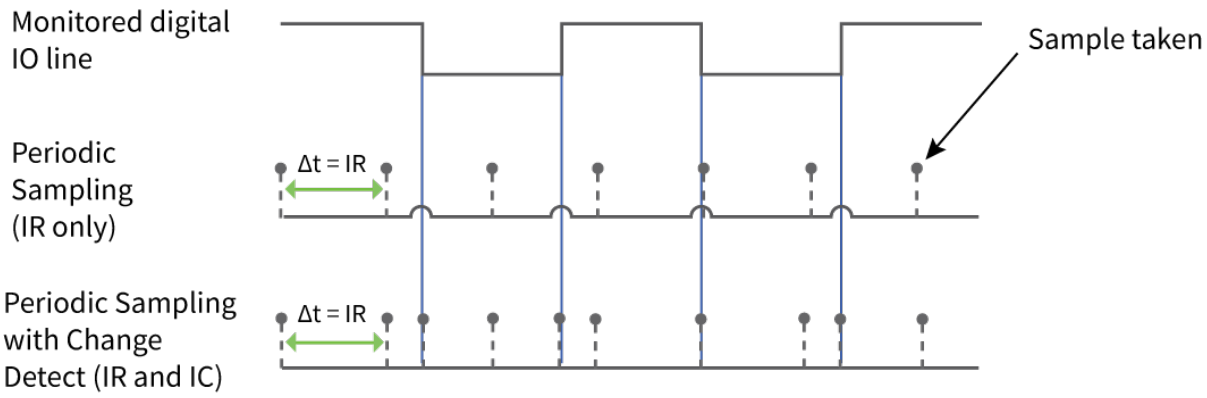
Output	Field	Description
00 04	Sample set 1	The first set of digital sample data that corresponds with the digital portion of the channel mask 0x0004 = 0000 0000 0000 0100b = DIO3 is high
01 28		Analog sample data for AD0
03 12		Analog sample data for AD1
00 00	Sample set 2	The second set of digital sample data 0x0004 = 0000 0000 0000 0000b = DIO3 is low
01 58		Second set of analog sample data for AD0
02 FE		Second set of analog sample data for AD1
00 04	Sample set 1	The third set of digital sample data 0x0004 = 0000 0000 0000 0100b = DIO3 is high
01 2A		Third set of analog sample data for AD0
03 A0		Third set of analog sample data for AD1
94	Checksum	Can safely be discarded on received frames

Digital I/O change detection

You can configure devices to transmit a data sample immediately whenever a monitored digital I/O pin changes state. IC (DIO Change Detect) is a bitmask that determines 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 on the monitored digital I/O line(s) using edge detection.

Change detection is only applicable to digital I/O pins that are configured as digital input (**3**) or digital output (**4** or **5**).

The figure below shows how I/O change detection can work in combination with Periodic I/O sampling to improve sampling accuracy. In the figure, the gray dashed lines with a dot on top represent samples taken from the monitored DIO line. The top graph shows only periodic IR samples, the bottom graph shows a combination of **IR** periodic samples and **IC** detected changes. In the top graph, the humps indicate that the sample was not taken at that exact moment and needed to wait for the next **IR** sample period.



Note Use caution when combining change detect sampling with sleep modes. **IC** only causes a sample to be generated if a state change occurs during a wake period. If the device is sleeping when the digital transition occurs, then no change is detected and an I/O sample is not generated.
Use periodic sampling with **IR** in conjunction with **IC** in this instance, since **IR** generates an I/O sample upon wakeup and ensures that the change is properly observed.

If you enable multiple samples by setting **IT** > **1**, any change detect that occurs causes all collected periodic samples to be sent immediately, then a separate **IC** sample is sent.

I/O line passing

Line passing allows you to affect the output pins of one device by sampling the I/O pins of another. To support line passing, you must configure a device to generate I/O sample data using periodic sampling (IR (Sample Rate)) and/or change detection (IC (DIO Change Detect)).

On the device that receives I/O samples, enable line passing setting IA (I/O Input Address) with the address of the device that has the appropriate inputs enabled. This effectively binds the outputs to a particular device's input. This does not affect the ability of the device to receive I/O line data from other devices—only its ability to update enabled outputs. Set **IA** to **0xFFFF** (broadcast address) to affect the output using input data from any device on the network.

Digital line passing

Digital I/O lines are mapped in pairs; pins configured as digital input on the transmitting device affect the corresponding digital output pin on the receiving device. For example, a device that samples **D5** as an input (3) only affects **D5** on the receiver if **D5** is configured as an output (4 or 5).

Each digital pin has an associated timeout value. When an I/O sample is received that affects a digital output pin, the pin returns to its configured state after the timeout period expires. For pins **D0** through **D9**, the associated timeout commands are T0 (D0 Timeout Timer) through T9 (D9 Output Timer). For pins **P0** through **P4**, the associated timeout commands are Q0 (P0 Output Timer) through Q2 (P2 Output Timer).

Digital line passing is only available on pins **D0** through **P3**. You cannot use UART and SPI pins for line passing.

Example: Digital line passing

A sampling XBee 3 802.15.4 RF Module is configured with the following settings:

AT command	Parameter value
D2 (DIO2/ADC2/TH_SPI_CLK Configuration)	3 (digital input)
IR (Sample Rate)	0x7D0 (2 seconds)
DH (Destination Address High)	0013A200
DL (Destination Address Low)	12345678

Every two seconds, an I/O sample is generated and sent to the address specified by **DH** and **DL**. The receiver is configured with the following settings:

AT command	Parameter value
D2 (DIO2/ADC2/TH_SPI_CLK Configuration)	5 (digital output low)
T2 (D2 Output Timeout Timer)	0x64 (10 seconds)
IA (I/O Input Address)	00103A20012345678

When this device receives an incoming I/O sample, if the source address matches the one set by **IA**, the device sets the output of **D2** to match the input of **D2** of the receiver. This output level holds for ten seconds before the pin returns to a digital output low state.

Analog line passing

Similar to digital line passing, analog line passing pairs the Analog I/O support of one device to a PWM output of another. There are two PWM output pins that can simulate the voltage measured by the ADC inputs. Be aware that ADC inputs are on different pins than the corresponding PWM outputs: AD0 corresponds to PWM0, and AD1 corresponds to PWM1. See Analog I/O support for the pinouts.

You can set the analog line passing timeout value with PT (PWM Output Timeout), which affects both PWM output pins. You can explicitly set a PWM output level using the M0 (PWM0 Duty Cycle) and M1 (PWM1 Duty Cycle) commands, when an I/O sample is received that affects a PWM output pin, it returns to its configured state after the **PT** timeout period expires.

Example: Analog line passing

A sampling device is configured with the following settings:

AT command	Parameter value
	2 (ADC input)
IR (Sample Rate)	0x7D0 (2 seconds)
DH (Destination Address High)	0013A200
DL (Destination Address Low)	12345678

Every two seconds, an I/O sample frame is generated and sent to the address specified by **DH** and **DL**. The receiver is configured with the following settings:

AT command	Parameter value
P0	2 (PWM output)
M0	0
PT	0x12C (30 seconds)
IA	0013A20087654321

When this device receives an incoming I/O sample, if the source address matches the one set by **IA**, the device sets the PWM output of **P0** to match the ADC input of **D0** of the receiver. This output level holds for thirty seconds before the pin returns to a digital output low state.

Output sample data

If a device receives an I/O sample whose address matches that set by IA (I/O Input Address), it triggers line passing. Line passing operates whether the receiving device is operating in API or Transparent mode.

By default, if the receiver is configured for API mode, it outputs the I/O sample frame in addition to affecting output pins. You can suppress the I/O sample frame output by setting IU (I/O Output Enable) to **0**. This only suppresses I/O samples that trigger line passing, a sample generated from a device whose address does not match the **IA** address is sent regardless of **IU**.

Output control

IO (Digital Output Level) controls the output levels of D0 (DIO0/ADC0/Commissioning Configuration) through D7 (DIO7/CTS Configuration) that are configured as output pins (either **4** or **5**). These values override the configured output levels of the pins until they are changed again (the pins do not automatically revert to their configured values after a timeout.)

You can use **IO** to trigger a sample on change detect.

I/O behavior during sleep

When the device sleeps (**SM !** = **0**) the I/O lines are optimized for a minimal sleep current.

Digital I/O lines

Digital I/O lines set as digital output high or low maintain those values during sleep. Disabled or input pins continue to be controlled by the **PR**/**PD** settings. Peripheral pins (with the exception of CTS) are set low during sleep and SPI pins are set high. Peripheral and SPI pins resume normal operation upon wake.

Digital I/O lines that have been set using I/O line passing hold their values during sleep, however the digital timeout timer (**T0** through **T9**, and **Q0** through **Q2**) are suspended during sleep and resume upon wake.

Analog and PWM I/O Lines

Lines configured as analog inputs or PWM output are not affected during sleep. PWM lines are shut down (set low) during sleep and resume normal operation upon wake.

PWM output pins set by analog line passing are shutdown during sleep and revert to their preset values (**M0** and **M1**) on wake. This happens regardless of whether the timeout has expired or not.

Networking

Networking terms

The following table describes some common terms we use when discussing networks.

Term	Definition
Association	Establishing membership between end devices and a coordinator.
Coordinator	A full-function device (FFD) that allows end devices to associate to it and can queue and deliver indirect messages.
End device	When in the same network as a coordinator. Devices that rely on a coordinator for synchronization and can be put into states of sleep for low-power applications.
PAN	Personal Area Network. A data communication network that includes one or more end devices and optionally a coordinator.

MAC Mode configuration

Medium Access Control (MAC) Mode configures two functions:

1. Enables or disables the use of a Digi header in the 802.15.4 RF packet.
   When the Digi header is enabled (**MM** = 0 or 3), duplicate packet detection is enabled as well as certain AT commands.
   MAC Modes 1 and 2 do not include a Digi header, which disables many features of the device. All data is strictly pass-through. These modes are intended to provide some compatibility with third-party 802.15.4 devices.
2. Enables or disables MAC acknowledgment request for unicast packets.
   When MAC ACK is enabled (**MM** = 0 or 2), transmitting devices send packets with an ACK request so receiving devices send an ACK back (acknowledgment of RF packet reception) to the transmitter. If the transmitting device does not receive the ACK, it re-sends the packet up to three times or until the ACK is received.
   MAC Modes 1 and 3 disable MAC acknowledgment. Transmitting devices send packets without an ACK request so receiving devices do not send an ACK back to the transmitter.
   Broadcast messages are always sent with the MAC ACK request disabled.

The following table summarizes the functionality.

Mode	Digi header	MAC ACK
0 (default)	X	X
1		
2		X
3	X	

The default value for the **MM** configuration parameter is 0 which enables both the Digi header and MAC acknowledgment.

Clear Channel Assessment (CCA)

Prior to transmitting a packet, the device performs a CCA (Clear Channel Assessment) on the channel to determine if the channel is available for transmission. The detected energy on the channel is compared with the **CA** (Clear Channel Assessment) parameter value. If the detected energy exceeds the **CA** parameter value, the device does not transmit the packet.

Also, the device inserts a delay before a transmission takes place. You can set this delay using the **RN** (Backoff Exponent) parameter. If you set **RN** to 0, there is no delay before the first CCA is performed. The **RN** parameter value is the equivalent of the "minBE" parameter in the 802.15.4 specification. The transmit sequence follows the 802.15.4 specification.

On a CCA failure, the device attempts to re-send the packet up to three additional times, meaning a total of four attempts.

CCA operations

CCA is a method of collision avoidance that is implemented by detecting the energy level on the transmission channel before starting the transmission. The CCA threshold (defined by the **CA** parameter) defines the energy level that it takes to block a transmission attempt. For example, if CCA is set to the default value of 0x32 (which is interpreted as -50 dBm) then energy detected above the -50 dBm level (for example -45 dBm) temporarily blocks a transmission attempt. But if the energy level is less than that (for example -70 dBm), the transmission is not blocked. The intent of this feature is to prevent simultaneous transmissions on the same channel.

You can disable CCA by setting **CA** to 0. Disabling CCA can improve latency in noisy environments, but it can also interfere with other devices that are operating on the same channel. Setting or changing **CA** to a non-zero value only takes effect upon boot. If you adjust the **CA** value, ensure that you write the setting to flash with WR (Write) and restart with an FR (Software Reset).

In the event that the energy level exceeds the threshold, the transmission is blocked for a random number of backoff periods. The number of backoff periods is defined by the following formula: random (2^n - 1), where *n* is defined by the **RN** parameter and increments after each CCA failure. When **RN** is set to its default value of 0, then 2^n -1 is 0, preventing any delay before the first energy detection on a new frame. However, n increments after each CCA failure, giving a greater range for the number of backoff periods between each energy detection cycle.

In the event that six energy detection cycles occur and each one detects too much energy, the application tries again 1 to 48 ms later. After the application retries are exhausted, then the transmission fails with a CCA error.

Whenever the MAC code reports a CCA failure, meaning that it performed six energy detection cycles with exponential random back-offs, and each one failed, the **EC** parameter is incremented. The **EC** parameter can be read at any time to find out how noisy the operating channel is. It continues to increment until it reaches its maximum value of 0xFFFF. To get new statistics, you can set **EC** back to 0.

Retries configuration

If you are operating in a MAC Mode that enables MAC ACK (**MM**=0 or **MM**=2), each RF packet will be sent with up to five 802.15.4 MAC-Layer retries, meaning six transmission attempts are performed. This is enabled by default and provides a minimal amount of reliability to unicast transmissions.

If you are operating in a MAC Mode that enables the Digi header (**MM**=0 or **MM**=3), then you can optionally include Application-Layer retries using the RR (XBee Retries) command. Each Application-Layer retry attempt to send the packet using five MAC-Layer retries. This can greatly increase the reliability of unicast transmissions with a risk of reduced throughput.

Transmit status based on MAC mode and XBee retries configurations

When working in API mode, a transmit request frame sent by the user is always answered with a transmit status frame sent by the device, if the frame ID is non-zero. A Frame ID of 0 specifies that the packet should be sent without an acknowledgment.

The following tables report the expected transmit status for unicast transmissions and the maximum number of MAC and application retries the device attempts.

The tables also report the transmit status reported when the device detects energy above the CCA threshold (when a CCA failure happens).

The following table applies in either of these cases:

- Digi header is disabled.
- Digi header is enabled and XBee Retries (**RR** parameter) is equal to 0 (default configuration).

Mac ACK Config	Destination reachable			Destination unreachable			CCA failure happened		
	TX status	Retries		TX status	Retries		TX status	Retries	
		MAC	App		MAC	App		MAC	App
Enabled	00 (Success)	up to 5	0	01 (No acknowledgment received)	5	0	02 (CCA failure)	5	0
Disabled	00 (Success)	0	0	00 (Success)	0	0	02 (CCA failure)	5	0

The following table applies when:

- Digi header is enabled and XBee Retries (**RR** parameter) > 0.

Mac ACK Config	Destination reachable			Destination unreachable			CCA failure happened		
	TX status	Retries		TX status	Retries		TX status	Retries	
		MAC	App		MAC	App		MAC	App
Enabled	00 (Success)	up to 5 per app retry	up to **RR** value	21 (Network ACK Failure)	5	**RR** value	02 (CCA failure)	5	**RR** value
Disabled	00 (Success)	0	up to **RR** value	21 (Network ACK Failure)	0	**RR** value	02 (CCA failure)	5	**RR** value

Addressing

Every RF data packet sent over-the-air contains a Source Address and Destination Address field in its header. The XBee 3 802.15.4 RF Module conforms to the 802.15.4 specification and supports both short 16-bit addresses and long 64-bit addresses. A unique 64-bit IEEE source address is assigned at the factory and can be read with the **SL** (Serial Number Low) and **SH** (Serial Number High) commands. A device uses its unique 64-bit address as its Source Address if its **MY** (16-bit Source Address) value is 0xFFFF or 0xFFFE. Since the default value for **MY** is **0**, devices use short source addressing by default.

Send packets to a specific device in Transparent API mode

To send a packet to a specific device using 64-bit addressing:

- Set the Destination Address (**DL** + **DH**) of the sender to match the Source Address (**SL** + **SH**) of the intended destination device.

To send a packet to a specific device using 16-bit addressing:

1. Set the **DL** parameter to equal the **MY** parameter of the intended destination device.
2. Set the **DH** parameter to 0.

Addressing modes

802.15.4 frames have a source address, a destination address, and a destination PAN ID in the over-the-air (OTA) frame. The source and destination addresses may be either long or short and the destination address may be either a unicast or a broadcast. The destination PAN ID is short and it may also be the broadcast PAN ID (**ID** is set to 0xFFFF).

In Transparent mode, the destination address is set by the **DH** and **DL** parameters, but, in API mode, it is set by the type of TX request used: 64-bit Transmit Request - 0x00 or 16-bit Transmit Request - 0x01 frames. In either Transparent mode or API mode, the destination PAN ID is set with the **ID** parameter, and the source address is set with the **MY** parameter if **MY** is less than **0xFFFE**, otherwise the source address is set with the device's serial number (**SH** and **SL**).

Broadcasts and unicasts

Broadcasts are identified by the 16-bit short address of **0xFFFF**. Any other destination address is considered a unicast and is a candidate for acknowledgments, if enabled.

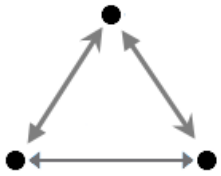
Broadcast PAN ID

The Broadcast PAN ID is also **0xFFFF**. Its effect is to traverse all PANs in the vicinity of a local device.

Short and long addresses

A short address is 16 bits and a long address is 64 bits. The short address is set with the **MY** parameter. If the short address is **0xFFFE**, then the address of the device is long and it is the serial number of the device as read by the **SH** and **SL** parameters.

Peer-to-peer networks



By default, XBee 3 802.15.4 RF Modules are configured to operate within a peer-to-peer network topology and therefore are not dependent upon master/slave relationships. Our peer-to-peer architecture features fast synchronization times and fast cold start times. This default configuration accommodates a wide range of RF data applications.

To form a peer-to-peer network, set each device to the same channel and PAN ID and configure either a unique short address (**MY**) for each device or set **MY** to **0xFFFF** to use the unique long addresses.

Master/slave networks

In a Master Slave network, there is a coordinator and one or more end devices. When end devices associate to the coordinator, they become members of that Personal Area Network (PAN). As such, they share the same channel and PAN ID. PAN IDs must be unique to prevent miscommunication between PANs. Depending on the **A1** and **A2** parameters, association may assist in automatically assigning the PAN ID and the channel. These parameters are specified below based on the network role (end device or coordinator).

End device association

End device association occurs if **CE** is **0** and **A1** has bit 2 set. See the following table and A1 (End Device Association).

Bit	Hex value	Meaning
0	0x01	Allow PAN ID reassignment
1	0x02	Allow channel reassignment
2	0x04	Auto association
3	0x08	Poll coordinator on pin wake

By default, **A1** is 0, which disables association and causes a device to operate in peer-to-peer mode. When bit 2 is set, the module becomes an end device and associates to a coordinator. This is done by sending out an active scan to detect beacons from nearby networks. The active scan iterates through each channel defined by **SC** and transmits a Beacon Request command to the broadcast address and the broadcast PAN ID. It then listens on that channel for beacons from any coordinator operating on that channel. Once that time expires, the active scan selects the next channel, repeating until all the channels defined by **SC** have been scanned.

If **A1** is **0x04** (bit 0 clear, bit 1 clear, and bit 2 set), then the active scan will reject all beacons that do not match both the configured PAN ID and the configured channel. This is the best way to join a particular coordinator.

If **A1** is **0x05** (bit 0 set, bit 1 clear, and bit 2 set), then the active scan will accept a beacon from any PAN ID, providing the channel matches. This is useful if the channel is known, but not the PAN ID.

If **A1** is **0x06** (bit 0 clear, bit 1 set, and bit 2 set), then the active scan will accept a beacon from any channel, providing the PAN ID matches. This is useful if the PAN ID is known, but not the channel.

If **A1** is **0x07** (bit 0 set, bit 1 set, and bit 2 set), then the active scan will accept a beacon from any PAN ID and from any channel. This is useful when the network does not matter, but the one with the best signal is desired.

Whenever multiple beacons are received that meet the criteria of the active scan, then the beacon with the best link quality is selected. This applies whether **A1** is **0x04**, **0x05**, **0x06**, or **0x07**.

Before the End Device joins a network, the Associate LED will be on solid. After it joins a network, the Associate LED will blink twice per second. You can also query the association status with AI (Association Indication) or by observing modem status frames when the end device is operating in API mode.

If association parameters are changed after the end device is associated, the end device will leave the network and re-join in accordance with the new configuration parameters.

After an end device successfully joins a network, the **DH** and **DL** parameters on the device are updated to point towards the address of the coordinator it associated with. This allows communication to the coordinator to occur automatically in Transparent mode, and ensures that indirect messaging poll requests are sent to the correct address—see Direct and indirect transmission.

Additionally, after associating, an end device has MY (16-bit Source Address) set to **0xFFFE**, indicating that the newly associated end device should use its 64-bit address. After associating, if you want a 16-bit address for the end device, set **MY** again.

Note **MY** is reset to **0xFFFE** if the end device needs to leave and re-associate with the coordinator.

If a coordinator changes channel or PAN ID, the end device is not informed of the change and indicates that it is still associated. You can set DA (Force Disassociation) on the end device to force it to leave the network and attempt to join again, validating that the end device can still communicate with the coordinator.

Coordinator association

A device becomes a coordinator and allows association if **CE** is 1 and **A2** has bit 2 set. See the following table and A2 (Coordinator Association).

Bit	Hex value	Meaning
0	0x01	Allow PAN ID reassignment
1	0x02	Allow channel reassignment
2	0x04	Allow association

By default, **A2** is 0, which prevents devices from associating to the coordinator. So, if **CE** is **1** and **A2** bit 2 is **0**, the device still creates a network, but end devices are unable to associate to it.

Note In this configuration, depending on the value of SP (Cyclic Sleep Period) the device might send messages indirectly—see Direct and indirect transmission.

If **A2** bit 2 is set, then joining is allowed after the coordinator forms a network.

If **A2** bit 0 is set, the coordinator performs an active scan. The active scan process sends a beacon request to the broadcast address (0xFFFF) and the broadcast PAN ID (0xFFFF) and listens for beacons responses. This process is repeated for each channel specified in **SC**.

If none of the beacons received during the active scan process match the ID parameter of the coordinator, then its ID parameter will be the PAN ID of the new network it forms. However, if a beacon response matches the PAN ID of the coordinator, the coordinator forms a PAN with a unique PAN ID.

If **A2** bit 0 is clear, then the coordinator forms a network on the PAN ID identified by the **ID** parameter, without regard to another network that might have the same PAN ID.

If **A2** bit 1 is set, the coordinator performs an energy scan, similar to the active scan. It will listen on each channel specified in the **SC** parameter. After the scan is complete, the channel with the least energy is selected to form the new network.

If **A2** bit 1 is clear, then no energy scan is performed and the **CH** parameter is used to select the channel of the new network.

If bits 0 and 1 of **A2** are both set, then an active scan is performed followed by an energy scan. However, the channels on which the active scan finds a coordinator are eliminated as possible channels for the energy scan, unless such an action would eliminate all channels. If beacons are found on all channels in the channel mask, then then the energy scan behaves the same as it would if beacons are not found on any of those channels. Therefore, the active scan will be performed on all channels in the channel mask. Then, an energy scan will be performed on the channels in the channel mask that did not find a coordinator.

Depending on the result of the active scan, the set of channels for the energy scan varies. If a PAN ID is found on all the channels in the channel mask, then the energy scan operates on all the channels in the channel mask. If at least one of the channels in the channel mask did not find a PAN ID, then the channels with PAN IDs are eliminated from consideration for the energy scan. After the energy scan completes, the channel with the least energy is selected for forming the new network.

Whenever **CE**, **ID**, **A2**, or **MY** changes, the coordinator will re-form the network. Any end devices associated to the coordinator prior to changing one of these parameters will lose association. For this reason, it is important not to change these parameters on a coordinator unless needed, or configure end devices to be flexible about what network they associate with the **A1** command.

Before the Coordinator forms a network, the Associate LED will be on solid. After it forms a network, the Associate LED will blink once per second.

Association indicators

There are two types of association indicators: Asynchronous device status messages, and on demand queries. Asynchronous device status messages occur whenever a change occurs and API mode is enabled. On demand queries occur when the **AI** command is issued, which can occur in Command mode, in API mode, or as a remote command.

Modem status messages

Not all device status messages are related with association, but for completeness all device status types reported by XBee 3 802.15.4 RF Module are listed in the following table.

Type	Meaning
0x00	Hardware reset.
0x01	Watchdog reset.
0x02	End device successfully associated with a coordinator.
0x03	End device disassociated from coordinator or coordinator failed to form a new network.

Type	Meaning
0x06	Coordinator formed a new network.
0x0D	Input voltage is too high, which limits RF power to **PL** = 3.

Association indicator status codes

The XBee 3 802.15.4 RF Module can potentially give any of the status codes in response to AI (Association Indication) in the following table.

Code	Meaning
0x00	Coordinator successfully started, End device successfully associated, or operating in peer to peer mode where no association is needed.
0x03	Active Scan found a PAN coordinator, but it is not currently accepting associations.
0x04	Active Scan found a PAN coordinator in a beacon-enabled network, which is not a supported feature.
0x05	Active Scan found a PAN, but the PAN ID does not match the configured PAN ID on the requesting end device and bit 0 of **A1** is not set to allow reassignment of PAN ID.
0x06	Active Scan found a PAN on a channel does not match the configured channel on the requesting end device and bit 1 of **A1** is not set to allow reassignment of the channel.
0x0C	Association request failed to get a response.
0x13	End device is disassociated or is in the process of disassociating.
0xFF	Initialization time; no association status has been determined yet.

Direct and indirect transmission

There are two methods to transmit data:

- Direct transmission: data is transmitted immediately to the Destination Address
- Indirect transmission: a packet is retained for a period of time and is only transmitted after the destination device (source address = destination address) requests the data.

Indirect transmissions can only occur on a device configured to be an indirect messaging coordinator. Indirect transmissions are useful to ensure packet delivery to a sleeping device. Indirect messaging allows messages to reliably be sent asynchronously to sleeping end devices, or operate like an incoming mailbox for a P2P network. A TX request can be made when the end device is sleeping and unable to receive RF data, and instead of being immediately send to an inoperative device, the packet is queued by the indirect messaging coordinator until the end device wakes or polls it for data.

Note that indirect messaging works best with association and end devices cyclically sleeping, but can be used in a P2P configuration by setting CE (Device Role) to **1** on the device that you want to hold the indirect messages and configuring the other device to poll correctly. In the context of indirect messaging, an end device refers not just to a device with A1 (End Device Association) set to associate but the target of an indirect message. Similarly, an indirect messaging coordinator does not have to allow association (A2 (Coordinator Association)) to send messages indirectly.

Configure an indirect messaging coordinator

A device becomes an indirect messaging coordinator once CE (Device Role) = **1** and SP (Cyclic Sleep Period) is not **0**. We recommend ensuring that **SP** and **ST** are set to the same values on the indirect messaging coordinator and end device, even if the indirect messaging coordinator is not configured to sleep. This is to allow the indirect messaging coordinator to send messages directly if it knows the end device is awake and sleeping cyclically.

If you are going to use a Master/Slave network with indirect messaging, ensure that the indirect messaging coordinator is also the network coordinator by allowing association (set bit 2 of A2 (Coordinator Association) to **1**).

Send indirect messages

To send an indirect message, ensure that the previous requirements are met and transmit normally. The indirect messaging coordinator queues the message until the end device requests data or the message is in the indirect queue for 2.5 times the value of **SP**. If 2.5 * **SP** is longer than 65 seconds, then 65 seconds is the limit the indirect message waits for a poll before it is discarded. This means that if the coordinator is sending data to the end device, the end device should poll the coordinator every 65 seconds to avoid losing data, regardless of the value of **SP**.

Ensure that the message is sent to the addressed specified by MY (16-bit Source Address) on the end device. If **MY** on the end device is **0xFFFF** or **0xFFFE**, then you must use the 64-bit address, otherwise use the value of **MY**. Even though an end device configured with a short address always receives direct transmissions destined to its 64-bit address, it will not receive an indirect message directed at its 64-bit address if it is configured to use a 16-bit address.

If the indirect messaging coordinator is operating in API mode, then after transmitting an indirect message the usual TX status frame (Extended Transmit Status - 0x8B or Transmit Status - 0x89) is not immediately generated by the device. If the end device polls for the data within the timeout (2.5 * **SP** or 65 seconds), then a TX status frame with status **0x00** (message sent) is sent. If the message is discarded due to the timeout expiring, the status frame is **0x03** (message purged).

After receiving a poll request and transmitting data to an end device, the indirect messaging coordinator sends all messages directly until **ST** time has elapsed. This is because after receiving RF data, the end device stays awake for **ST** time if configured in Cyclic Sleep mode (SM = 4). After **ST** time has elapsed, messages are sent indirectly again.

The Coordinator currently is able to retain up to five indirect messages.

Receive indirect messages

End devices must poll the indirect messaging coordinator in order to receive indirect messages.

There are three ways to generate a poll request:

- End devices using cyclic sleep automatically send a poll to the coordinator when they wake up unless **SO** bit 0 is set.
- End devices using pin sleep may be configured to send a poll on a pin wakeup by setting bit 3 of **A1**.
- Use FP (Force Poll) to manually send a poll to the coordinator. In Transparent mode, the poll request is not sent until the command is exited.

The poll is sent to the address located in **DH** and **DL**, so ensure that they are set to match the coordinator's source addressing mode. If the end device (**A1** bit 2 set) has associated with a coordinator (**A2** bit 2 set and **CE** = **1**), then **DH** and **DL** are automatically set to the correct values. If

you use indirect messaging in a P2P network, **DH** and **DL** have to be set manually on the end device to point towards the indirect messaging coordinator.

It is more difficult to use indirect messaging with pin sleep than with cyclic sleep because the end device must wake up periodically to poll for the data from the coordinator. Otherwise, the coordinator discards the data after **SP***2.5 time, or 65 seconds, whichever is smaller. It is also important to keep the pin woke device awake for **ST** time after receiving indirect messages, otherwise the coordinator could attempt to transmit directly while the end device is asleep, and the transmission will fail. For this reason we recommend only using indirect messaging with cyclic sleep.

Encryption

The XBee 3 802.15.4 RF Module supports AES 128-bit encryption. 128-bit encryption refers to the length of the encryption key entered with the **KY** command (128 bits = 16 bytes). The 802.15.4 protocol specifies eight security modes, enumerated as shown in the following table.

Level	Name	Encrypted?	Length of message integrity check	Packet length overhead
0	N/A	No	0 (no check)	0
1	MIC-32	No	4	9
2	MIC-64	No	8	13
3	MIC-128	No	16	21
4	ENC	Yes	0 (no check)	5
5	ENC-MIC-32	Yes	4	9
6	ENC-MIC-64	Yes	8	13
7	ENC-MIC-128	Yes	16	21

The XBee 3 802.15.4 RF Module only supports security levels 0 and 4. It does not support message integrity checks. **EE 0** selects security level 0 and **EE 1** selects security level 4. When using encryption, all devices in the network must use the same 16-byte encryption key for valid data to get through. Mismatched keys will corrupt the data output on the receiving device. Mismatched **EE** parameters will prevent the receiving device from outputting received data.

Working from a maximum packet size of 116 bytes, encryption affects the maximum payload as shown in the following table.

Factor	Effect on maximum payload	Comment
Compatibility mode	Force to 95	If **C8** bit 0 is set, all packets are limited to 95 bytes, regardless of other factors listed below. This is how the Legacy 802.15.4 module (S1 hardware) functions.
Packet overhead	Reduce by 5	This penalty for enabling encryption is unavoidable due to the 802.15.4 protocol.

Factor	Effect on maximum payload	Comment
Source address	Reduce by 6	This penalty is unavoidable because the 802.15.4 requires encrypted packets to be sent with a long source address, even if a short address would otherwise be used.
Destination address	Reduce by 6	This penalty only applies if sending to a long address rather than a short address.
App header	Reduce by 4	The app header for encryption is 4 bytes long. This penalty only applies if **MM** = 0 or 3.

Because of the two mandatory reductions when using encryption, no packet can exceed 116 - (5+6) =105 bytes. The other options may further reduce the maximum payload to 101 bytes, 99 bytes, or 95 bytes.

When operating in API mode and not using encryption, if the source address is long, the receiving device outputs an RX Indicator (0x80) frame for received data. But, if the source address is short, the receiving device outputs a Receive Packet (0x81) frame for received data. These same rules apply for encryption if **MM** is **0** or **3**. This is possible because the four-byte encryption App header includes the short address of the sender and the long received address is not used for API output. If encryption is enabled with **MM** of **1** or **2**, then no App header exists, the source address is always long, and the receiving device in legacy API mode (**AP** = **2**) always outputs a Description.

Maximum payload

The absolute maximum payload size for an 802.15.4 packet is 116 bytes. Depending on module configuration, the actual maximum payload size will be reduced.

If you attempt to send an API packet with a larger payload than specified, the device responds with a Transmit Status frame (0x89 and 0x8B) with the Status field set to 74 (Data payload too large). When operating in transparent mode, if you attempt to send data larger than the maximum payload size, the data will be packetized and sent as multiple over-the-air transmissions. For more information, see Serial-to-RF packetization.

Maximum payload rules

1. If you enable transmit compatibility (**C8**) with the Legacy 802.15.4 module (S1 hardware):
   - There is a fixed maximum payload of 100 bytes
   - The rest of the rules do not apply. They apply only when you disable transmit compatibility with the Legacy 802.15.4 module.
2. The maximum achievable payload is 116 bytes. This is achieved when:
   - Not using encryption.
   - Not using the application header (**MM** is set to 1 or 2).
   - Using the short source address.
   - Using the short destination address.

3. If you are using the application header, the maximum achievable payload is reduced by:
   - 2 bytes if not using encryption (**EE** = **0**)
   - 4 bytes if using encryption (**EE** = **1**)
4. If you are using the long source address (**MY** = **0xFFFE**), the maximum achievable payload is reduced by 6 bytes (size of long address (8) - size of short address (2) = 6).
5. If you are using encryption, the source addresses are promoted to long source addresses, so the maximum achievable payload is reduced by 6 bytes.
6. If you are using the long destination address, the maximum achievable payload is reduced by 6 bytes (the difference between the 8 bytes required for a long address and the 2 bytes required for a short address).
7. If you are using encryption, the maximum achievable payload is reduced by 5 bytes.

Note You can query NP (Maximum Packet Payload Bytes) to determine the maximum achievable payload size based on current parameters. **NP** always assumes a long destination address will be used.

Maximum payload summary tables

The following table indicates the maximum payload when using transmit compatibility with Legacy 802.15.4 modules (S1 hardware).

Encryption	
Enabled	**Disabled**
95 B	100 B

The following table indicates the maximum payload when using the application header and not using encryption. Increment the maximum payload in 2 bytes if you are not using the application header.

	Destination address	
Source address	Short	Long
Short	114 B	108 B
Long	108 B	102 B

The following table indicates the maximum payload when using the application header and using encryption. Increment the maximum payload in 4 bytes if you are not using the application header.

	Destination address	
Source address	Short	Long
Short	101 B	95 B
Long	101 B	95 B

Work with Legacy devices

The Legacy 802.15.4 device (S1 hardware) transmits packets one by one. It does not transmit a packet until it receives all expected acknowledgments of the previous packet or the timeout expires.

The XBee/XBee-PRO S2C 802.15.4 and XBee 3 802.15.4 RF Modules enhance transmission by implementing a transmission queue that allows the device to transmit to several devices at the same time. Broadcast transmissions are performed in parallel with the unicast transmissions.

This enhancement in the XBee/XBee-PRO S2C 802.15.4 and XBee 3 802.15.4 RF Modules can produce problematic behavior under certain conditions if the receiver is a Legacy 802.15.4 module (S1 hardware).

The conditions are:

- The sender is an XBee 3 802.15.4 RF Module, and the receiver is a Legacy 802.15.4 module.
- The sender has the Digi header enabled (**MM** = 0 or 3) and **RR** (XBee Retries) > 0.
- The sender sends broadcast and unicast messages at the same time to the Legacy 802.15.4 module without waiting for the transmission status of the previous packet.

The effect is:

- The receiver may display duplicate packets.

The solution is:

- Set bit 0 of the **C8** (802.15.4 compatibility) parameter to 1 to enable TX compatibility mode in the XBee 3 802.15.4 RF Module. This eliminates the transmission queue to avoid sending to multiple addresses simultaneously. It also limits the packet size to the levels of the Legacy 802.15.4 module.

For information on the specific differences between an XBee 3 and Legacy 802.15.4 devices, refer to the Digi XBee 3 802.15.4 Migration Guide.

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, devices include features to aid in placing devices, configuring devices, and network diagnostics.

Remote configuration commands

When running in API mode, the firmware has provisions to send configuration commands to remote devices using Remote AT Command Request - 0x17. You can use this frame to send commands to a remote device to read or set command parameters.

CAUTION! It is important to set the short address to 0xFFFE when sending to a long address. Any other value causes the long address to be ignored. This is particularly problematic in the case where nodes are set up with default addresses of 0 and the 16-bit address is erroneously left at 0. In that case, even with a correct long address the remote command goes out to all devices with the default short address of 0, potentially resulting in harmful consequences, depending on the command.

Send a remote command

To send a remote command populate the Remote AT Command Request frame (0x17) with:

1. The 64-bit address of the remote device.
2. The correct command options value.
3. The command and parameter data (optional). If (and *only* if) all nodes in the PAN have unique short addresses, then remote configuration commands can be sent to 16-bit short addresses by setting the short address in the API frame for Remote AT commands. In that case, the 64-bit address is unused and does not matter.

Apply changes on remote devices

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. Set the Apply Changes option bit in the Remote AT Command Request frame (0x17).
2. Issue an **AC** (Apply Changes) command to the remote device.
3. Issue a **WR** + **FR** command to the remote device to save changes and reset the device.

Remote command responses

If the remote device receives a Remote AT Command Request (0x17 frame type), the remote sends an AT Command Response (0x88 frame type) back to the device that sent the remote command. The AT command response indicates the status of the command (success, or reason for failure), and in the case of a command query, it includes the parameter value.

The device that sends a remote command will not receive a remote command response frame if the frame ID in the remote command request is set to 0 , indicating that the request is sent without acknowledgment.

Node discovery

Node discovery has three variations as shown in the following table:

Commands	Syntax	Description
ND (Network Discover)	**ND**	Seeks to discover all nodes in the network (on the current PAN ID).
ND (Network Discover)	**ND**<NI String>	Seeks to discover if a particular node named <NI String> is found in the network.
DN (Discover Node)	**DN**<NI String>	Sets **DH**/**DL** to point to the address (64-bit or 16-bit depending on the **MY** value of the matching node) of the node whose <NI String> matches.

The node discovery command (without an **NI** string designated) sends out a broadcast to every node in the PAN ID. Each node in the PAN sends a response back to the requesting node after a jittered time delay to ensure reliable delivery.

About node discovery

The node discovery command (without an NI string designated) sends out a broadcast to every node in the PAN ID. Each node in the PAN sends a response back to the requesting node.

When the node discovery command is issued in AT command mode, all other AT commands are inhibited until the node discovery command times out, as specified by the **NT** parameter. After the timeout, an extra CRLF is output to the terminal window, indicating that new AT commands can be entered. This is the behavior whether or not there were any nodes that responded to the broadcast.

When the node discovery command is issued in API mode, the behavior is the same except that the response is output in API mode. If no nodes respond, there will be no responses at all to the node discover command. The requesting node is not able to process a new AT command until **NT** times out.

Node discovery in compatibility mode

Node discovery (without an **NI** string parameter) in compatibility mode operates the same in compatibility mode as it does outside of compatibility mode with one minor exception:

If **C8** bit 1 is set and if requesting node is operating in API mode and if no responses are received by the time **NT** times out, then an API AT command response of OK (API frame type 0x88) is sent out the serial port rather than giving no response at all, which would happen if **C8** bit 1 is not set.

Directed node discovery

The directed node discovery command (**ND** with an **NI** string parameter) sends out a broadcast to find a node in the network with a matching **NI** string. If such a node exists, it sends a response with its information back to the requesting node.

In Transparent mode, the requesting node will output an extra CRLF following the response from the designated node and the command will terminate, being ready to accept a new AT command. In the event that the requested node does not exist or is too slow to respond, the requesting node outputs an **ERROR** response after **NT** expires.

In API mode, the response from the requesting node will be output in API mode and the command will terminate immediately. If no response comes from the requested node, the requesting node outputs an error response in API mode after **NT** expires.

Directed node discovery in compatibility mode

The behavior of the Legacy 802.15.4 module (S1 hardware) varies with the default behavior described above for the directed node discovery command. The Legacy module does not complete the command until **NT** expires, even if the requested node responds immediately. After **NT** expires, it gives a successful response, even if the requested node did not respond. To enable this behavior to be equivalent to the Legacy 802.15.4 module, set bit 1 of the **C8** parameter.

Destination Node

DN (Discover Node) with anNI (Node Identifier) string parameter sends out a broadcast containing the **NI** string being requested. The responding node with a matching **NI** string sends its information back to the requesting node. The local node then sets **DH**/**DL** to match the address of the responding node. As soon as this response occurs, the command terminates successfully. If operating in Command mode, an **OK** string is output and Command mode exits. In API mode another AT command may be entered.

If an **NI** string parameter is not provided, the **DN** command terminates immediately with an error. If a node with the given **NI** string doesn't respond, the **DN** command terminates with an error after **NT** times out.

Unlike **ND** (with or without an **NI** string), **DN** does not cause the information from the responding node to be output; rather it simply sets **DH**/**DL** to the address of the responding node. If the responding node has a short address, then **DH**/**DL** is set to that short address (with **DH** at 0 and **DL** set to the value of **MY**). If the responding node has a long address (**MY** is **0xFFFE**), then **DH**/**DL** are set to the **SH**/**SL** of the responding node.

Sleep support

Sleep is implemented to support installations where a mains power source is not available and a battery is required. In order to increase battery life, the device sleeps, which means it stops operating. It can be woken by a timer expiration or a pin.

Sleep modes

Sleep modes enable the device to enter states of low-power consumption when not in use. To enter Sleep mode, the following conditions must be met:

- A valid sleep mode is selected via **SM** (**SM** = **1**, **4**, **5**, or **6**)
- DTR/SLEEP_RQ (TH pin 9/SMT pin 10) is asserted (when **SM** = **1** or **5**)
- The device is idle (no data transmission or reception) for the amount of time defined by ST (Cyclic Sleep Wake Time) (when **SM** = **4** or **5**)

The following table shows the sleep mode configurations.

Sleep mode	Description
SM 0	No sleep
SM 1	Pin sleep
SM 4	Cyclic sleep
SM 5	Cyclic sleep with pin wake-up
SM 6	MicroPython sleep (with optional pin wake). For complete details see the *Digi MicroPython Programming Guide*.

Pin Sleep mode (SM = 1)

Pin Sleep mode minimizes quiescent power (power consumed when in a state of rest or inactivity). In order to use Pin Sleep mode, configure D8 (DIO8/DTR/SLP_Request Configuration) (TH pin 9/SMT pin 10) for DTR/SLEEP_RQ input (**D8** = **1**). This mode is voltage level-activated; when SLEEP_RQ is asserted, the device finishes any transmit or receive activities, enters Idle mode, and then enters a state of sleep. The device does not respond to either serial or RF activity while in pin sleep.

To wake a sleeping device operating in Pin Sleep mode, de-assert DTR/SLEEP_RQ. The device wakes when SLEEP_RQ is de-asserted and is ready to transmit or receive when the CTS line is low. When waking the device, the pin must be de-asserted at least two 'byte times' after CTS goes low. This assures that there is time for the data to enter the DI buffer.

Devices with SPI functionality can use the SPI_SSEL pin instead of **D8** for pin sleep control. If **D8** = **0** and **P7** = **1**, SPI_SSEL takes the place of DTR/SLEEP_RQ and functions as described above. In order to use SPI_SSEL for sleep control while communicating on the UART, the other SPI pins must be disabled (**P5**, **P6**, and **P8** set to **0**). See Low power operation for information on using SPI_SSEL for sleep control while communicating over SPI.

Cyclic Sleep mode (SM = 4)

The Cyclic Sleep modes allow devices to periodically check for RF data. When the **SM** parameter is set to **4**, the XBee 3 802.15.4 RF Module is configured to sleep, then wakes once per cycle to check for data from a coordinator. The Cyclic Sleep Remote sends a poll request to the coordinator at a specific interval set by the **SP** (Cyclic Sleep Period) parameter. The coordinator transmits any queued data addressed to that specific remote upon receiving the poll request.

If the coordinator does not respond with queued data and no UART activity is detected, the device will immediately sleep. If it detects any activity (RF or UART), then the device wakes for **ST** time. You can also set **SO** bit 8 to force the device to always wake for the full **ST** time.

ON_SLEEP goes high and CTS goes low each time the remote wakes, allowing for communication initiated by the remote host if desired.

Cyclic Sleep with Pin Wake-up mode (SM = 5)

Use this mode to wake a sleeping remote device through either the RF interface or by asserting (low) DTR/SLEEP_RQ for event-driven communications. The cyclic sleep mode works as described previously with the addition of a pin-controlled wake-up at the remote device.

The DTR/SLEEP_RQ pin is level-triggered. The device wakes when a low is detected then sets CTS low as soon as it is ready to transmit or receive. The device stays awake as long as DTR/SLEEP_RQ is low; once DTR/SLEEP_RQ goes high the device returns to cyclic sleep operation. If DTR/SLEEP_RQ is momentarily pulsed low, the minimum wake time is ST (Cyclic Sleep Wake Time) even if DTR/SLEEP_RQ is low for less time.

Once awake, any activity resets the ST (Cyclic Sleep Wake Time) timer, so the device goes back to sleep only after there is no RF activity for the duration of the timer.

MicroPython sleep with optional pin wake (SM = 6)

The MicroPython sleep option allows a user's MicroPython program to exclusively control the device's sleep operation (with optional pin wake). For full details refer to the *Digi MicroPython Programming Guide*.

Sleep parameters

The following AT commands are associated with the sleep modes. See the linked commands for the parameter's description, range and default values.

- SM (Sleep Mode)
- SP (Cyclic Sleep Period)
- ST (Cyclic Sleep Wake Time)
- DP (Disassociated Cyclic Sleep Period)
- SO (Sleep Options)

Sleep pins

The following table describes the three external device pins associated with sleep. See the *XBee 3 RF Module Hardware Reference Manual* for the pinout of your device.

Pin name	Description
DTR/SLEEP_RQ	For **SM** = **1**, high puts the device to sleep and low wakes it up. For **SM** = **5**, a high to low transition wakes the device until the pin transitions back to a high state.
SPI_SSEL	Alternative SLEEP_RQ line for devices operating in SPI. See Low power operation for further information.
CTS	If **D7** = **1**, high indicates that the device is asleep and low indicates that it is awake and ready to receive serial data.
ON_SLEEP	Low indicates that the device is asleep and high indicates that it is awake.

Sleep conditions

Since instructions stop executing while the device is sleeping, it is important to avoid sleeping when the device has work to do. For example, the device will not sleep if any of the following are true:

1. The device is operating in Command mode, or in the process of getting into Command mode with the **+++** sequence.
2. The device is processing AT commands from API mode
3. The device is processing remote AT commands
4. Something is queued to the serial port and that data is not blocked by $\overline{\text{RTS}}$ flow control

If each of the above conditions are false, then sleep may still be blocked in these cases:

1. Enough time has not expired since the device has awakened.
    a. If the device is operating in pin sleep, the amount of time needed for one character to be received on the UART is enough time.
    b. If the device is operating in cyclic sleep, enough time is defined by a timer. The duration of that timer is:
        i. defined by **ST** if in **SM** 5 mode and it is awakened by a pin
        ii. 30 ms to allow enough time for a poll and a poll response
        iii. 750 ms to allow enough time for association, in case that needs to happen
    c. In addition, the wake time is extended by an additional **ST** time when new OTA data or serial data is received.
2. Sleep Request pin is not asserted when operating in pin sleep mode
3. Data is waiting to be sent OTA.

AT commands

Networking commands

Configure the basic 802.15.4 network settings. All devices on the network must have matching network settings to communicate.

CH (Operating Channel)

The operating channel devices use to transmit and receive data.

In order for devices to communicate with each other, they must share the same channel number. A network can use different channels to prevent devices in one network from listening to the transmissions of another and to reduce interference.

The command uses IEEE 802.15.4 channel numbers.

Parameter range

0xB - 0x1A

Default

0xC (channel 12)

ID (Extended PAN ID)

The device's PAN (Personal Area Network) identifier. PAN IDs allows for the logical separation of multiple networks that share the same RF channel.

In order for devices to communicate, they must be configured with the same PAN ID and channel.

Setting **ID** to **0xFFFF** indicates a global transmission for all PANs. It does not indicate a global receive.

Parameter range

0 - 0xFFFF

Default

0x3332

MM (MAC Mode)

Use the **MM** command to specify the operating MAC Mode; for more information see MAC Mode configuration.

The MAC Mode serves two purposes:

- Enable/disable the use of a Digi header, which enables advanced features.
- Enable/disable MAC-Layer acknowledgments.

The default configuration includes a Digi-specific header to every RF packet. This header includes information that enables advanced features:

- Network discovery support [ND (Network Discover) and DN (Discover Node)]
- Application-layer retries [RR (XBee Retries)]
- Duplicate packet detection [RR (XBee Retries)]
- Remote AT command support [Remote AT Command Request - 0x17]

The presence of the Digi header prevents interoperability with third-party devices. When the Digi header is disabled, encrypted data that is not valid is sent out of the UART and not filtered out. The Digi header can be disabled by setting **MM** to **1** or **2**.

When **MM** is set to **1** or **3**, MAC-layer retries are disabled.

Parameter range

0 - 3

Parameter	Configuration	ACKs
0	Digi mode	With ACKs
1	802.15.4	No ACKs
2	802.15.4	With ACKs
3	Digi mode	No ACKs

Default

0

C8 (Compatibility Options)

Sets the operational compatibility with the legacy 802.15.4 device (S1 hardware). This parameter should only be set when operating in a mixed network that contains XBee Series 1 devices.

Parameter range

0 - 3

Bit field:

Unused bits must be set to **0**. These bits may be logically OR'ed together:

Bit	Meaning	Setting	Description
0[1]	TX compatibility	0	Transmissions are optimized as follows: 1. Maximum transmission size is affected by multiple factors (**MM**, **MY**, **DH**, **DL**, and **EE**). See Maximum payload rules. In the best case, with no app header, short source and destination addresses, and no encryption, the maximum transmission size is 116 bytes. 2. Multiple messages can be present simultaneously on the active queue, providing they are all destined for different addresses. This improves performance.
		1	Transmissions operate like the Legacy 802.15.4 module, which means the following: 1. Maximum transmission size is 95 bytes for encrypted packets and 100 bytes for un-encrypted packets. These maximum transmission sizes are not adjusted upward for short addresses or for lack of an APP header. 2. Only one transmission message can be active at a time, even if other messages in the queue would go to a different destination address.
1	Node Discovery compatibility	0	Node discovery operates like other XBee devices and not like the Legacy 802.15.4 module. This means the following: 1. A directed **ND** request terminates after the single response arrives. This allows the device to process other commands without waiting for the **NT** to time out. 2. The device outputs an error response to the directed **ND** request if no response occurs within the time out.
		1	The module operates like the Legacy 802.15.4 module, which has the following effect: 1. When the expected response arrives, the command remains active until **NT** times out. (**NT** defaults to 2.5 seconds.) This prevents the device from processing any other AT command, even if the desired response occurs immediately. 2. When the timeout occurs, the command silently terminates and indicates success, whether or not a response occurred within the **NT** timeout.

Default

0x00

[1]This bit does not typically need to be set. However, when the XBee 3 802.15.4 RF Module is streaming broadcasts in transparent mode to a Legacy 802.15.4 module (S1 hardware), and **RR** > 0, set this bit to avoid a watchdog reset on the Legacy 802.15.4 module.

Discovery commands

Network Discovery and corresponding discovery options.

Network discovery can only be performed if the Digi header is enabled via the **MM** command.

NI (Node Identifier)

The node identifier is a user-defined name or description of the device. Use this string with network discovery commands in order to easily identify devices on the network.

Use the ND (Network Discover) command with this string as an argument to filter network discovery results.

Use the DN (Discover Node) command with this string as an argument to resolve the 64-bit address of a node with a matching **NI** string.

Parameter range

A string of case-sensitive ASCII printable characters from 1 to 20 bytes in length. A carriage return or a comma automatically ends the command.

Default

0x20 (an ASCII space character)

DD (Device Type Identifier)

Stores the Digi device type identifier value. Use this value to differentiate between multiple types of devices (for example, sensors or lights).

This command can optionally be included in network discovery responses by setting bit 1 of **NO**.

Parameter range

0 - 0xFFFFFFFF

Default

0x130000

NT (Node Discover Timeout)

Sets the amount of time a base node waits for responses from other nodes when using the ND (Network Discover) and DN (Discover Node) 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.

Parameter range

0x1 - 0xFC (x 100 ms)

Default

0x19 (2.5 seconds)

NO (Network Discovery Options)

Use **NO** to suppress or include a self-response to **ND** (Node Discover) commands. When **NO** bit 1 is set, a device performing a Node Discover includes a response entry for itself.

Bit field:

Unused bits must be set to **0**. These bits may be logically OR'ed together:

Parameter range

0 - 1

Default

0x0

ND (Network Discover)

This command reports the following information after a jittered time delay. Node discover response when issued in Command mode:

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.)

A second carriage return indicates the network discovery timeout (**NT**) has expired.

When operating in API mode and a Network Discovery is issued as a 0x08 or 0x09 frame, the response contains binary data except for the NI string in the following format:

2 bytes for Short Source Address

4 bytes for Upper Long Address

4 bytes for Lower Long Address

1 byte for the signal strength in -dBm (two's complement representation)

NULL-terminated string for NI (Node Identifier) value (maximum 20 bytes without NULL terminator)

Each device that responds to the request will generate a separate Description.

Broadcast an **ND** command to the network. If the command includes an optional node identifier string parameter, only those devices with a matching **NI** string respond without a random offset delay. If the command does not include a node identifier string parameter, all devices respond with a random offset delay. If there are no matching devices to the string identifier parameter, the command returns an “ERROR” if the device is in Transparent mode.

The **NT** setting determines the maximum timeout (13 seconds by default), this value is sent along with the discovery broadcast and determines the random delay the remote nodes use to prevent the responses from colliding.

For more information about the options that affect the behavior of the **ND** command, see NO (Network Discovery Options).

WARNING! If the **NT** setting is small relative to the number of devices on the network, responses may be lost due to channel congestion. Regardless of the **NT** setting, because the random offset only mitigates transmission collisions, getting responses from all devices in the network is not guaranteed.

The **ND** command cannot be issued from within MicroPython or over BLE.

Parameter range

20-byte printable ASCII string (optional)

Default

N/A

DN (Discover Node)

Resolves an **NI** (Node identifier) string to a physical address (case sensitive).

The **DN** command cannot be issued from within MicroPython or over BLE.

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 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, then Command mode does not exit.

When **DN** is sent as a local AT Command API frame:

1. The receiving device returns the 16-bit network and 64-bit extended addresses in an API Command Response frame.
2. If there is no response from a module within (**NT*** 100) milliseconds or you do not specify a parameter (by leaving it blank), the receiving device returns an ERROR message. In the case of an ERROR, the device does not exit Command mode. Set the radius of the **DN** command using the **BH** command.

When **DN** is sent as a local Local AT Command Request - 0x08:

1. The receiving device returns a success response in a Description.
2. If there is no response from a module within (**NT** * 100) milliseconds or you do not specify a parameter (by leaving it blank), the receiving device returns an ERROR message.

Parameter range

20-byte ASCII string

Default

N/A

AS (Active Scan)

Sends a Beacon Request to a Broadcast address (**0xFFFF**) and Broadcast PAN (**0xFFFF**) on every channel in the scan channel mask—SC (Scan Channels). Active Scan can only be performed locally and

returns an ERROR if attempted remotely.

The **AS** command cannot be issued from within MicroPython or over BLE.

A PanDescriptor is created and returned for every Beacon received from the scan. Each PanDescriptor contains the following information:

CoordAddress (**SH** + **SL** parameters)<CR>

Note If **MY** on the coordinator is set less than 0xFFFF, the **MY** value is displayed.

CoordPanID (**ID** parameter)<CR>

CoordAddrMode <CR>

0x02 = 16-bit Short Address

0x03 = 64-bit Long Address

Channel (**CH** parameter) <CR>

SecurityUse<CR>

ACLEntry<CR>

SecurityFailure<CR>

SuperFrameSpec<CR> (2 bytes):

bit 15 - Association Permitted (MSB)

bit 14 - PAN Coordinator

bit 13 - Reserved

bit 12 - Battery Life Extension

bits 8-11 - Final CAP Slot

bits 4-7 - Superframe Order

bits 0-3 - Beacon Order

GtsPermit<CR>

RSSI<CR> (- RSSI is returned as -dBm)

TimeStamp<CR> (3 bytes)

<CR> (A carriage return indicates the end of the PanDescriptor)

The Active Scan returns one PanDescriptor response per discovered network. Each PanDescriptor has a trailing carriage return <CR> to indicate the end of the frame. The sequence of PanDescriptors has a final trailing carriage return (three <CR> in sequence indicate the end of the active scan).

If using API Mode, no <CR>'s are returned and a separate response frame is generated for each PanDescriptor. For more information, see Operate in API mode. If no PANs are discovered during the scan, only one carriage return is printed.

The **AS** command cannot be issued from within MicroPython or over BLE.

Before a device is associated to a network (**AI** != **0**), it will continuously perform an active scan in the background, searching for a valid network to join. While this is occurring, you cannot manually perform an active scan using the **AS** command. You can bypass this restriction by setting **DJ** to **1**. This will disable joining and halt the background active scans.

Parameter range

N/A

Default

N/A

Coordinator/End Device configuration commands

The following commands configure the device for a master/slave 802.15.4 network.

CE (Device Role)

The default configuration for an XBee 3 802.15.4 network to operate in a Peer-to-Peer configuration. In a Peer-to-Peer network, every device must have a preconfigured network PAN ID and RF Channel in order to communicate.

In a Peer-to-Peer network, setting **CE** to **1** configures the device to act as an indirect messaging coordinator if **SP** is non-zero.

The XBee 3 802.15.4 network can also be configured for Master/Slave operation. This is enabled by setting bit 2 of the A1 (End Device Association) or A2 (Coordinator Association) commands and setting **CE** accordingly. The configuration of the of the master/slave network is determined by the **A1** and **A2** commands. A network coordinator can also act as an indirect messaging coordinator if **SP** is non-zero.

End Devices configured for cyclic sleep will use DP (Disassociated Cyclic Sleep Period) instead of SP (Cyclic Sleep Period) until it associates with a coordinator.

Parameter range

0 - 1

Parameter	Description
0	Peer-to-Peer / Network End Device
1	Network/Indirect Messaging Coordinator

Default

0

Note If **CE** = **1** and **SP** is not **0**, then all messages are sent indirectly. See Direct and indirect transmission for more information.

A1 (End Device Association)

Sets or displays the End Device association options. These options are only applicable when configured as an End Device by setting CE (Device Role) to **0**.

Bit 2 must be set before other options are enabled.

Parameter range

0 - 0x0F (bit field)

Bit field:

Bit	Meaning	Setting	Description
0	Allow PanId reassignment	0	Only associates with Coordinator operating on PAN ID that matches device ID.
		1	May associate with Coordinator operating on any PAN ID.

Bit	Meaning	Setting	Description
1	Allow Channel reassignment	0	Only associates with Coordinator operating on matching **CH** channel setting.
		1	May associate with Coordinator operating on any channel defined in the SC (Scan Channels) mask.
2	Auto Associate	0	Peer-to-Peer operation - Device will not attempt association.
		1	Network End Device - Device attempts association until success. If configured for cyclic sleep, DP (Disassociated Cyclic Sleep Period) is used instead of SP (Cyclic Sleep Period) until the device associates.
3	Poll coordinator on pin wake	0	Pin Wake does not poll the Coordinator for indirect (pending) data.
		1	Pin Wake sends Poll Request to Coordinator to extract any pending data.
4 - 7	Reserved		

Default

0

A2 (Coordinator Association)

Sets or displays the Coordinator association options. These options are only applicable when configured as a coordinator by setting CE (Device Role) to **1**. Bit 2 must be set before other options are enabled.

Parameter range

0 - 7 (bit field)

Bit field:

Bit	Meaning	Setting	Description
0	Allow Pan ID reassignment	0	Coordinator will not perform Active Scan to locate available PAN ID. It operates on ID (PAN ID).
		1	Coordinator performs an Active Scan to determine an available **ID** (PAN ID). If a PAN ID conflict is found, the **ID** parameter will change.
1	Allow Channel reassignment	0	Coordinator will not perform Energy Scan to determine free channel. It operates on the channel determined by the **CH** parameter.
		1	Coordinator performs an Energy Scan to find the quietest channel out of the channels to be scanned determined by the **SC** parameter. The Coordinator then operates on that channel.
2	Allow Association	0	Peer-to-Peer - Will not allow any devices to associate to it.
		1	Network Coordinator - Allows devices to associate to it.
3 - 7	Reserved		

Default

0

SC (Scan Channels)

Sets or displays the list of channels to scan for all Active and Energy Scans as a bit field. This affects scans initiated in AS (Active Scan) and ED (Energy Detect) commands in Command mode and during End Device Association and Coordinator startup.

Parameter range

0 - 0xFFFF (bit field)

Note A parameter of **0** will scan on the current channel configured by **CH**.

Bit field mask:

Bit	IEEE 802.15.4 Channel	Frequency (GHz)
0	11 (0x0B)	2.405
1	12 (0x0C)	2.410
2	13 (0x0D)	2.415
3	14 (0x0E)	2.420
4	15 (0x0F)	2.425
5	16 (0x10)	2.430
6	17 (0x11)	2.435
7	18 (0x12)	2.440
8	19 (0x13)	2.445
9	20 (0x14)	2.450
10	21 (0x15)	2.455
11	22 (0x16)	2.460
12	23 (0x17)	2.465
13	24 (0x18)	2.470
14	25 (0x19)	2.475
15	26 (0x1A)	2.480

Note Avoid channel 26 if possible, as the output power is capped at +8 dBm on the Pro variant.

Default

0xFFFF

SD (Scan Duration)

Sets or displays the scan duration exponent.

Scan Time is measured as:

([# of channels to scan] * (2 ^**SD**) * 15.36 ms) + (38 ms * [# of channels to scan]) + 20 ms

Use the **SC** (Scan Channels) command to set the number of channels to scan.

Example

The following table shows the results for a thirteen channel scan.

SD setting	Time
0x0	0.18 s
0x2	0.74 s
0x4	2.95 s
0x6	11.80 s
0x8	47.19 s
0xA	3.15 min
0xB	12.58 min
0xE	50.33 min

Parameter range

0 - 0x0F (exponent)

Default

4

DA (Force Disassociation)

Causes the End Device to immediately disassociate from a Coordinator (if associated) and re-attempt to associate.

Parameter range

N/A

Default

N/A

AI (Association Indication)

Reads the Association status code to monitor association progress.

The following table provides the status codes and their meanings.

Status code	Meaning
0x00	Coordinator successfully started, End device successfully associated, or operating in peer to peer mode where no association is needed.
0x03	Active Scan found a PAN coordinator, but it isn't currently accepting associations.
0x05	Active Scan found a PAN, but the PAN ID doesn't match the configured PAN ID on the requesting end device and bit 0 of **A1** is not set to allow reassignment of PAN ID.
0x06	Active Scan found a PAN on a channel that does not match the configured channel on the requesting end device and bit 1 of **A1** is not set to allow reassignment of the channel.
0x0C	Association request failed to get a response.
0x13	End device is disassociated or is in the process of disassociating.
0xFF	Initialization time; no association status has been determined yet.

Parameter range

0 - 0xFF [read-only]

Default

N/A

802.15.4 Addressing commands

The following commands affect how outgoing 802.15.4 transmissions are addressed and configured.

SH (Serial Number High)

Displays the upper 32 bits of the unique IEEE 64-bit extended address assigned to the XBee in the factory.

This value is read-only and it never changes.

Parameter range

0x0013A200 - 0x0013A2FF [read-only]

Default

Set in the factory

SL (Serial Number Low)

Displays the lower 32 bits of the unique IEEE 64-bit RF extended address assigned to the XBee in the factory.

This value is read-only and it never changes.

Parameter range

0 - 0xFFFFFFFF [read-only]

Default

Set in the factory

MY (16-bit Source Address)

Sets or displays the device's 16-bit source address. Set **MY** = **0xFFFE** to disable reception of packets with 16-bit addresses. To maintain compatibility with older products, **0xFFFF** is also acceptable to disable the reception of packets with 16-bit addresses. When configured in this way, the 64-bit long source address (**SH**+**SL**) is used for outgoing messages.

Regardless of **MY**, messages addressed to the 64-bit long address of the device are always delivered.

Parameter range

0 - 0xFFFF

Default

0

DH (Destination Address High)

Set or read the upper 32 bits of the 64-bit destination address.

When you combine **DH** with **DL**, it defines the 64-bit destination address that the device uses for outgoing data transmissions in Transparent mode (**AP** = **0**) and I/O sampling. This destination address corresponds to the serial number (**SH** + **SL**) of the target device.

To transmit using a 16-bit address, set **DH** to 0 and **DL** less than 0xFFFF. When associating to a coordinator as an end device (**CE** = **0**, **A1** | 0x04), the destination address is automatically set to address the coordinator.

Reserved 802.15.4 network addresses:

- **0x000000000000FFFF** is a broadcast address (**DH** = **0**, **DL** = **0xFFFF**).

Parameter range

0 - 0xFFFFFFFF

Default

0

DL (Destination Address Low)

Set or read the lower 32 bits of the 64-bit destination address.

When you combine **DH** with **DL**, it defines the 64-bit destination address that the device uses for outgoing data transmissions in Transparent mode (**AP** = **0**) and I/O sampling. This destination address corresponds to the serial number (**SH** + **SL**) of the target device.

To transmit using a 16-bit address, set **DH** to **0** and **DL** less than **0xFFFF**. When associating to a coordinator as an end device (**CE** = **0**, **A1** | 0x04), the destination address is automatically set to address the coordinator.

Reserved 802.15.4 network addresses:

- 0x000000000000FFFF is a broadcast address (**DH** = **0**, **DL** = **0xFFFF**).

Parameter range

0 - 0xFFFFFFFF

Default

0

RR (XBee Retries)

Set or reads the number of application-layer retries the device executes. Application-layer retries are only enabled if a Digi header is present via the **MM** command.

Every transmitted unicast transmission uses up to five MAC-Layer retries (if enabled via the **MM** command). If **RR** > 0, a failed unicast transmission will be attempted **RR** times (each application-layer retry will exhaust the five MAC-layer retries).

When transmitting a broadcast message, if **RR** = 0, only one packet is broadcast. If **RR** is > 0, then **RR** + 2 packets are sent on each broadcast. No acknowledgments are returned on a broadcast.

The **RR** value does not need to be set on all devices for retries to work. If retries are enabled, the transmitting device sets a bit in the Digi RF Packet header that requests the receiving device to send an ACK. Each device retry can potentially result in the MAC sending the packet six times (one try plus five retries).

Parameter range

0 - 6

Default

0

TO (Transmit Options)

A bitfield that configures the advanced options used for outgoing data transmissions from a device operating in Transparent mode (**AP** = **0**).

When operating in API mode, if the Transmit Options field in the API frame is 0, the **TO** parameter value will be used instead.

Parameter range

0 - 0xFF

Bit field:

Unused bits must be set to **0**. These bits may be logically OR'ed together:

Bit	Meaning
0	Disable MAC acknowledgments (retries) for unicast traffic.
2	Send to broadcast PAN ID.
4	Send data securely—requires secure session be established with destination. Enabling this bit will reduce maximum payload size by 4 bytes.

Default

0

NP (Maximum Packet Payload Bytes)

Reads the maximum number of RF payload bytes that you can typically send in a transmission based on current parameter settings. Some options may impact maximum payload size that are not captured by the **NP** value.

See Maximum payload for more information.

NP is based on multiple factors including the length of the source address, the length of the destination address, the length of the APP header, and whether or not encryption is enabled.

For the purposes of this command, it always assumes a long destination address. This means that if you select a short destination address, you will be able to send up to **NP** + 6 bytes in a single packet.

Note **NP** returns a hexadecimal value. For example, if **NP** returns 0x66, this is equivalent to 102 bytes.

Parameter range

0 - 0xFF [read-only]

Default

N/A

Security commands

The following commands enable and control the encryption used for RF transmissions.

EE (Encryption Enable)

Enables or disables 128-bit Advanced Encryption Standard (AES) encryption of RD data transmissions.

The firmware uses the 802.15.4 Default Security protocol and uses AES encryption with a 128-bit key. AES encryption dictates that all devices in the network use the same key, and that the maximum RF packet size is 95 bytes if Tx compatibility is enabled (you set bit 0 of **C8**). If **C8**, bit 0 is not set, see Maximum payload.

When encryption is enabled, the device always uses its 64-bit long address as the source address for RF packets. This does not affect how the **MY** (Source Address), **DH** (Destination Address High) and **DL** (Destination Address Low) parameters work.

If **MM** (MAC Mode) is set to 1 or 2 and **AP** (API Enable) parameter > 0:

With encryption enabled and a 16-bit short address set, receiving devices can only issue RX (Receive) 64-bit indicators. This is not an issue when **MM** = **0** or **3**.

If a device with a non-matching key detects RF data, but has an incorrect key:

When encryption is enabled, non-encrypted RF packets received are rejected and are not sent out the UART.

Parameter range

0 - 1

Parameter	Description
0	Encryption Disabled
1	Encryption Enabled

Default

0

KY (AES Encryption Key)

Sets the 128-bit network security key value that the device uses for encryption and decryption.

This command is write-only and cannot be read. 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 entire payload of the packet is encrypted using the key and the CRC is computed across the ciphertext.

Parameter range

128-bit value (up to 16 bytes)

Default

0

DM (Disable Features)

Bit field:

Unused bits must be set to **0**. These bits may be logically OR'ed together:

A bit field mask that you can use to enable or disable specific features.

If disabling device functionality for security purposes, we recommend that you also enable secure remote configuration to prevent features from being re-enabled remotely.

Bit	Description
0	Reserved
1	Reserved
2	Disable firmware over-the-air (FOTA) updates. When set to 1, the device cannot act as a FOTA update client. FOTA File System access is protected with FK (File System Public Key). **Note** Serial firmware updates are always possible via the bootloader.
3	Disable SRP authentication on the client side of the connection.
4	Disable SRP authentication on the server side of the connection.

Parameter range

0, 4 - 0x1F (bit field)

Default

0

US (OTA Upgrade Server)

Specifies the 64-bit address of the server the device should use for OTA upgrades.

- 0: Accept OTA upgrades from any device
- 0x1-0xFFFFFFFFFFFFFFFE: Only accept OTA upgrades from a server with the given 64-bit address
- 0xFFFFFFFFFFFFFFFF: Reserved

Note If this parameter is not **0**, packets from the OTA server must be sent with 64-bit addressing. This is done by setting **MY** to **0xFFFE**.

Parameter range

0 - 0xFFFFFFFFFFFFFFFE

Default

0

Secure Session commands

These are the AT commands that enable Secure Session.

SA (Secure Access)

The Secure Access Options bit-field defines the feature set(s) intended to be secure against unauthorized access. The XBee 3 802.15.4 RF Module should establish a secure session in order to access functionality defined by the feature set(s) on the local device.

A password must be set using the Secure Session Salt and Verifier before access is secured.

Parameter range

0 - 0x1F (up to 0xFFFF)

Bit field

Unused bits must be set to **0**. These bits may be logically OR'ed together:

Bit	Description
0	Reserved
1	Remote AT Commands When set to **1** and if a password has been set, the device will not respond to insecure Remote AT Command requests (API Frame 0x17) but still can send insecure Remote AT Commands.
2	Serial Data When set to **1**, the device will not emit any serial data that was sent insecurely. This functionality applies to devices that are configured for Transparent mode, but in this instance, only the SRP server would be **AP** = **0**, the client would still have to send the Secure Session Control - 0x2E via API mode. The server will also not emit any 0x90 or 0x91 frames when this bit is set. **Note** On 802.15.4 insecure 0x80 frames will also not be emitted.

Default

0

*S (Secure Session Salt)

The Secure Remote Password (SRP) Salt is a 32-bit number used to create an encrypted password for the XBee 3 802.15.4 RF Module. The ***S** command contains the salt value in the salt/verifier pair used

for secure session authentication.

Parameter range

0-FFFFFFFF

Default

0

*V, *W, *X, *Y (Secure Session Verifier)

The secure session verifier is a 128-byte value used together with *S (Secure Session Salt) for secure session authentication. The ***V**, ***W**, ***X**, and ***Y** commands each contain 32 bytes of the secure session verifier: ***V** contains bytes 0 - 31, ***W** bytes 32 - 63, ***X** bytes 54 - 95, and ***Y** bytes 96 - 127.

Parameter range

Each command can be any 32-byte value: 0-FFFFFFFF

Default

0

RF interfacing commands

The following AT commands affect the 2.4 GHz 802.15.4 RF interface of the device.

PL (TX Power Level)

Sets or displays the power level at which the device transmits conducted power for 802.15.4 traffic.

Note If operating on channel 26 (**CH** = **0x1A**), output power will be capped and cannot exceed 8 dBm regardless of the **PL** setting.

Parameter range

0 - 4

Parameter	XBee non-PRO	XBee 3 PRO
0	-5 dBm	-5 dBm
1	-1 dBm	+3 dBm
2	+2 dBm	+8 dBm
3	+5 dBm	+15 dBm
4	+8 dBm	+19 dBm

Default

4

PP (Output Power in dBm)

Display the operating output power based on the current configuration (channel and **PL** setting). The values returned are in dBm, with negative values represented in two's complement; for example: -5 dBm = 0xFB.

Parameter range

0 - 0xFF [read-only]

Default

N/A

CA (CCA Threshold)

Defines the Clear Channel Assessment (CCA) threshold. Prior to transmitting a packet, the device performs a CCA to detect energy on the channel. If the device detects energy above the CCA threshold, it will not transmit the packet.

The **CA** parameter is measured in units of -dBm. The CCA threshold is set upon device initialization, any change to the CCA threshold must be written to flash with the **WR** command and the module reset (power cycle or **FR** command) before the new threshold is observed.

You can set **CA** to 0 to disable CCA; this can improve latency but may cause interference with other 2.4 GHz devices when transmitting.

Parameter range

0 (disabled), 0x28 - 0x64 (-dBm)

Default

0x41

RN (Random Delay Slots)

Defines the minimum value of the back-off exponent in the CSMA-CA algorithm. The Carrier Sense Multiple Access - Collision Avoidance (CSMA-CA) algorithm was engineered for collision avoidance (random delays are inserted to prevent data loss caused by data collisions.

If **RN** = 0, there is no delay between a request to transmit and the first iteration of CSMA-CA.

Unlike CSMA-CD, which reacts to network transmissions after collisions have been detected, CSMA-CA acts to prevent data collisions before they occur. As soon as a device receives a packet that is to be transmitted, it checks if the channel is clear (no other device is transmitting). If the channel is clear, the packet is sent over-the-air. If the channel is not clear, the device waits for a randomly selected period of time, then checks again to see if the channel is clear. After a time, the process ends and the data is lost.

Parameter range

0 - 5 (exponent)

Default

0

MAC diagnostics commands

The following commands provide Media Access Control diagnostic information.

DB (Last Packet RSSI)

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.

If the XBee 3 802.15.4 RF Module has been reset and has not yet received a packet, **DB** reports **0**.

This value is volatile (the value does not persist in the device's memory after a power-up sequence).

Parameter range

0 - 0xFF [read-only]

Default

N/A

EA (ACK Failures)

The number of unicast transmissions that time out awaiting a MAC ACK. This can be up to **RR** +1 timeouts per unicast when **RR** > 0.

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 unsigned value, append a hexadecimal parameter to the command.

This value is volatile (the value does not persist in the device's memory after a power-up sequence).

Parameter range

0 - 0xFFFF

Default

0x0

EC (CCA Failures)

Sets or displays the number of frames that were blocked and not sent due to CCA failures or receptions in progress. If CCA is disabled (**CA** is **0**), then this count only increments for frames that are blocked due to receive in progress. When this count reaches its maximum value of **0xFFFF**, it stops counting.

You can reset **EC** to **0** (or any other value) at any time to make it easier to track errors. This value is volatile (the value does not persist in the device's memory after a power-up sequence).

Parameter range

0 - 0xFFFF

Default

0x0

ED (Energy Detect)

Starts an energy detect scan. The device loops through all the available channels and 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.

ED accepts a parameter value but it will not affect the scan duration or results. **ED** cannot be issued from within MicroPython or over BLE.

Parameter range

0 - 0xFF

Default

N/A

Sleep settings commands

The following commands enable and configure the low power sleep modes of the device.

SM (Sleep Mode)

Sets or displays the sleep mode of the device.

By default, Sleep Modes are disabled (**SM** = 0) and the device remains in Idle/Receive mode. When in this state, the device is constantly ready to respond to either serial or RF activity.

When operating in Pin Sleep (**SM** = 1), D8 (DIO8/DTR/SLP_Request Configuration) must be set as a peripheral (**D8**=1) in order for the device to sleep.

Parameter range

0 - 5

Parameter	Description
0	No sleep (disabled)
1	Pin sleep
2	Reserved
3	Reserved
4	Cyclic Sleep Remote
5	Cyclic Sleep Remote with pin wakeup
6	MicroPython sleep (with optional pin wake). For complete details see the *Digi MicroPython Programming Guide*.

Default

0

SP (Cyclic Sleep Period)

Sets and reads the duration of time that a remote device sleeps. After the cyclic sleep period is over, the device wakes and checks for data. If data is not present, the device goes back to sleep. The

maximum sleep period is 4 hours (**SP** = 0x15F900).

The **SP** parameter is only valid if you configure the end device to operate in Cyclic Sleep (**SM** = 4-5). Coordinator and End Device **SP** values should always be equal.

To send direct messages on a coordinator, set **SP** = 0. If the device is a coordinator (CE (Device Role) = **1**) and **SP** is not **0**, the device sends all transmissions indirectly, meaning end devices have to poll the coordinator to receive data—FP (Force Poll) or using cyclic sleep.

End Device: **SP** determines the sleep period for cyclic sleeping remotes. The maximum sleep period is 4 hours (0x15F900).

Coordinator: If non-zero, **SP** determines the time to hold an indirect message before discarding it. A Coordinator discards indirect messages after a period of (2.5 * **SP**, or 65 seconds, whichever is smaller).

Parameter range

0x0 - 0x15F900 (x 10 ms) (4 hours)

Default

0x0

ST (Cyclic Sleep Wake Time)

Sets or displays the wake time of the device.

The **ST** parameter is only valid for end devices configured with Cyclic Sleep settings (**SM** = 4 - 5) and for coordinators. Upon waking the device polls for queued indirect messages and UART data. If it does not detect activity, the device immediately sleeps. The device only stays awake for **ST** time if RF or UART activity is detected upon wakeup or bit 8 of SO (Sleep Options) is set to **1**.

Coordinator and End Device **ST** values must be equal.

Parameter range

0x1 - 0x36EE80 (x 1 ms)

Default

0x7D0 (2 seconds)

DP (Disassociated Cyclic Sleep Period)

Sets or displays the sleep period for cyclic sleeping remotes that are configured for Association but that are not associated to a Coordinator. For example, if a device is configured to associate and is configured as a Cyclic Sleep remote, but does not find a Coordinator, it sleeps for **DP** time before reattempting association.

Parameter range

1 - 0x15F900 (x 10 ms)

Default

0x3E8 (10 seconds)

SN (Number of Sleep Periods)

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 if no RF data is

waiting for the end device. This command allows a host application to sleep for an extended time if no RF data is present.

Parameter range

1 - 0xFFFF

Default

1

SO (Sleep Options)

A bitfield that contains advanced sleep options that do not have dedicated AT commands.

Parameter range

0 - 0x103

Bit field:

Unused bits must be set to **0**. These bits may be logically OR'ed together:

Bit	Setting	Meaning	Description
0	0	Normal operations	A device configured for cyclic sleep will poll for data on waking
	1	Disable wakeup poll	A device configured for cyclic sleep will not poll for data on waking
1	0	Normal operations	A device configured in a sleep mode with ADC/DIO sampling enabled will automatically perform a sampling on wakeup
	1	Suppress sample on wakeup	A device configured in a sleep mode with ADC/DIO sampling enabled will not automatically sample on wakeup
8	0	Normal operations	A device configured for cyclic sleep will wake only momentarily after the expiration of **SP**
	1	Always wake for **ST** time	A device configured for cyclic sleep will always remain awake for **ST** time before returning to sleep

Default

0

FP (Force Poll)

The **FP** command is deferred until changes are applied. This prevents indirect messages from arriving at the end device while it is operating in Command mode.

Parameter range

N/A

Default

N/A

MicroPython commands

The following commands relate to using MicroPython on the XBee 3 802.15.4 RF Module.

PS (Python Startup)

Sets whether or not the XBee 3 802.15.4 RF Module runs the stored Python code at startup.

Range

0 - 1

Parameter	Description
0	Do not run stored Python code at startup.
1	Run stored Python code at startup.

Default

0

PY (MicroPython Command)

Interact with the XBee 3 802.15.4 RF Module using MicroPython. **PY** is a command with sub-commands. These sub-commands are arguments to **PY**.

PYB (Bundled Code Report)

You can store compiled code in flash using the **os.bundle()** function in the MicroPython REPL; refer to the *Digi MicroPython Programming Guide*. The **PYB** sub-command reports details of the bundled code. In Command mode, it returns two lines of text, for example:

```
bytecode: 619 bytes (hash=0x0900DBCE)
compiled: 2017-05-09T15:49:44
```

The messages are:

- **bytecode**: the size of bytecode stored in flash and its 32-bit hash. A size of **0** indicates that there is no stored code.
- **compiled**: a compilation timestamp. A timestamp of **2000-01-01T00:00:00** indicates that the clock was not set during compilation.

In API mode, **PYB** returns three 32-bit big-endian values:

- bytecode size
- bytecode hash
- timestamp as seconds since 2000-01-01T00:00:00

PYE (Erase Bundled Code)

PYE interrupts any running code, erases any bundled code and then does a soft-reboot on the MicroPython subsystem.

PYV (Version Report)

Report the MicroPython version.

PY^ (Interrupt Program)

Sends **KeyboardInterrupt** to MicroPython. This is useful if there is a runaway MicroPython program and you have filled the stdin buffer. You can enter Command mode (**+++**) and send **ATPY^** to interrupt the program.

Default

N/A

File System commands

To access the file system, enter Command mode and use the following commands. All commands block the AT command processor until completed and only work from Command mode; they are not valid for API mode or MicroPython's **xbee.atcmd()** method. Commands are case-insensitive as are file and directory names. Optional parameters are shown in square brackets (**[]**).

FS (File System)

FS is a command with sub-commands. These sub-commands are arguments to **FS**.

Error responses

If a command succeeds it returns information such as the name of the current working directory or a list of files, or **OK** if there is no information to report. If it fails, you see a detailed error message instead of the typical **ERROR** response for a failing AT command. The response is a named error code and a textual description of the error.

Note The exact content of error messages may change in the future. All errors start with a upper case **E**, followed by one or more uppercase letters and digits, a space, and an description of the error. If writing your own AT command parsing code, you can determine if an **FS** command response is an error by checking if the first letter of the response is upper case **E**.

FS (File System)

When sent without any parameters, **FS** prints a list of supported commands.

FS PWD

Prints the current working directory, which always starts with **/** and defaults to **/flash** at startup.

FS CD *directory*

Changes the current working directory to **directory**. Prints the current working directory or an error if unable to change to **directory**.

FS MD *directory*

Creates the directory **directory**. Prints **OK** if successful or an error if unable to create the requested directory.

FS LS [*directory*]

Lists files and directories in the specified directory. The **directory** parameter is optional and defaults to a period (**.**), which represents the current directory. The list ends with a blank line.

Entries start with zero or more spaces, followed by file size or the string **<DIR>** for directories, then a single space character and the name of the entry. Directory names end with a forward slash (**/**) to differentiate them from files.

```
<DIR> ./
<DIR> ../
<DIR> lib/
    32 test.txt
```

FS PUT filename

Starts a YMODEM receive on the XBee Smart Modem, storing the received file to **filename** and ignoring the filename that appears in block 0 of the YMODEM transfer. The XBee Smart Modem sends a prompt (**Receiving file with YMODEM...**) when it is ready to receive, at which point you should initiate a YMODEM send in your terminal emulator.

If the command is incorrect, the reply will be an error as described in Error responses.

FS HASH *filename*

Print a SHA-256 hash of a file to allow for verification against a local copy of the file. On Windows, you can generate a SHA-256 hash of a file with the command **certutil -hashfile test.txt SHA256**. On Mac and Linux use **shasum -b -a 256 test.txt**.

FS GET *filename*

Starts a YMODEM send of filename on the XBee device. When it is ready to send, the XBee Smart Modem sends a prompt: (**Sending file with YMODEM...**). When the prompt is sent, you should initiate a YMODEM receive in your terminal emulator.

If the command is incorrect, the reply will be an error as described in Error responses.

FS RM *file_or_directory*

Removes the file or empty directory specified by **file_or_directory**. This command fails with an error if **file_or_directory** does not exist, is not empty, refers to the current working directory or one of its parents.

Note Removing a file only reclaims space if the file removed is placed last in the file system. Deleted data that is contiguous with the last deleted file is also reclaimed. Directories are only reclaimed if all directories in that particular block of memory are deleted and found at the end of the file system. Use the **ATFS INFO FULL** command to see where in the file system files and directories are placed.

FS INFO

Report on the size of the filesystem, showing bytes in use, available, marked bad and total. The report ends with a blank line, as with most multi-line AT command output. Example output:

```
204800 used
 695296 free
     0 bad
 900096 total
```

FS INFO FULL

Reports every file and directory in the order they are placed in the file system along with the amount of space they take up individually. Also reports deleted space as well as unused directory slots. Example output:

```
128 /flash./
128 /flash/lib./
128 /flash/directory./
1664 [unused dir slot(s)]
```

```
2048 /flash/file1.txt.
2048 [deleted space]
2048 /flash/directory/file2.txt
```

FS FORMAT confirm

Formats the file system, leaving it with a default directory structure. Pass the word **confirm** as the first parameter to confirm the format. The XBee Smart Modem responds with **Formatting...** when the format starts, and will print **OK** followed by a carriage return when it finishes.

FK (File System Public Key)

Configures the device's File System Public Key (all 65-bytes must be entered, including any leading zeros).

You must set **FK** locally via Command Mode or 0x08 or 0x09 API frames. You cannot set the public key remotely.

The 65-byte public key is required to verify that the file system that is downloaded over-the-air is a valid XBee 3 file system compatible with the 802.15.4 firmware.

For further information, refer to Set the public key on the XBee 3 device.

Parameter range

A valid 65-byte ECDSA public key—all 65-bytes must be entered, including any leading zeros.

Other accepted parameters:

0 = Clear the public key

1 = Returns the upper 48 bytes of the public key

2 = Returns the lower 17 bytes of the public key

Default

0

Note The Default value of **0** indicates that no public key has been set and hence, all file system updates will be rejected.

Bluetooth Low Energy (BLE) commands

The following AT commands are BLE commands.

BT (Bluetooth Enable)

BT enables or disables the Bluetooth functionality.

Note When Bluetooth is enabled, the XBee 3 802.15.4 RF Module cannot be in Sleep mode. If the device is configured to allow Sleep mode and you enable Bluetooth, the XBee 3 802.15.4 RF Module will not enter sleep.

WARNING! RF data loss may be encountered when BLE is enabled due to the PHY switching between RF and BLE. We highly recommended that you enable retries and multi-transmit—via the **RR** and **MT** commands—when BLE is enabled.

Parameter range

Parameter	Description
0	Bluetooth functionality is disabled.
1	Bluetooth functionality is enabled.

Default

0

BL (Bluetooth MAC Address)

BL reports the EUI-48 Bluetooth device address. Due to standard XBee AT Command processing, leading zeroes are not included in the response when in Command mode.

Parameter range

N/A

Default

N/A

BI (Bluetooth Identifier)

A human-friendly name for the device. This is the name that will appear in bluetooth advertisement messages.

If set to default (ASCII space character), the bluetooth indicator will display as **XBee3 802.15.4**.

If using XBee Mobile, adjustments to the filter options will be needed if this value is populated.

Parameter range

A string of case-sensitive ASCII printable characters from 1 to 22 bytes in length.

Default

0x20 (an ASCII space character)

BP (Bluetooth Power)

Sets the power level for Bluetooth Advertisements. All other BLE transmissions are sent at 8 dBm.

Parameter range

Parameter	Description
0	-20 dBm
1	-10 dBm
2	0 dBm
3	8 dBm

Default

3 = 8 dBm

$S (SRP Salt)

Note You should only use this command if you have already configured a password on the XBee device and the salt corresponds to the password.

The Secure Remote Password (SRP) Salt is a 32-bit number used to create an encrypted password for the XBee 3 802.15.4 RF Module. Use the **$S** command in conjunction with the **$V**, **$W**, **$X**, and **$Y** verifiers. Together, the command and the verifiers authenticate the client for the BLE API Service without storing the XBee password on the XBee 3 802.15.4 RF Module.

Configure the salt in the **$S** command. In the **$V**, **$W**, **$X**, and **$Y** verifiers, you specify the 128-byte verifier value, where each command represents 32 bytes of the total 128-byte verifier value.

Note The XBee 3 802.15.4 RF Module does not allow for **0** to be valid salt. If the value is **0**, SRP is disabled and you are not able to authenticate using Bluetooth.

Parameter range

0 - FFFFFFFF

Default

0

$V, $W, $X, $Y commands (SRP Salt verifier)

Use the **$V**, **$W**, **$X**, and **$Y** verifiers in conjunction with $S (SRP Salt) to create an encrypted password for the XBee 3 802.15.4 RF Module. Together, **$S** and the verifiers authenticate the client for the BLE API Service without storing the XBee password on the XBee device.

Configure the salt with the **$S** command. In the **$V**, **$W**, **$X**, and **$Y** verifiers, you specify the 128-byte verifier value, where each command represents 32 bytes of the total 128-byte verifier value.

Parameter range

0 - FFFFFFFF

Default

0

API configuration commmands

The following commands affect how API mode operates.

AP (API Enable)

Set or read the API mode setting. The device can format the RF packets it receives into API frames and sends them out the serial port.

For more information, see Serial modes.

When you enable API, you must format the serial data as API frames because Transparent operating mode is disabled.

Parameter range

0 - 2

Parameter	Description
0	API disabled (operate in Transparent mode)
1	API enabled
2	API enabled (with escaped control characters)
4	API enabled (operate in Micropython mode)

Default

0

AO (API Output Options)

Configure the serial output and legacy I/O sampling options for received API frames. This parameter is only applicable when the device is operating in API mode (**AP** = **1** or **2**). **AO** also affects how I/O samples are gathered and transmitted. For detailed information on how I/O sampling is handled, see Legacy support.

For new designs, we recommend **AO** = **0**. This provides API compatibility with DigiMesh and Zigbee applications and allows for all 15 I/O lines to be sampled (**D0** through **P4**). Incoming serial data packets will be emitted as Transmit Request - 0x10. All outgoing I/O samples will be sent as I/O Sample Indicator - 0x92.

When **AO** is set to **2**, I/O samples are transmitted in a legacy format that are compatible with legacy S1 and S2C 802.15.4 XBee devcies. As a result, only 9 I/O lines are available (**D0** through **D8**) for sampling. Incoming data packets will be emitted as either 0x81 or 0x82 frames depending on the addressing scheme of the sender. All outgoing I/O samples will be sent as 0x82 or 0x83 frames depending on the addressing scheme of the sender.

Parameter range

0 - 2

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.
2	Legacy 802.15.4 API Indicator - 0x80/0x81. Also restricts the Digital Input sampling to **D0** through **D8** and allows for OTA compatibility with legacy S1 and S2C devices.

Default

2

AZ (Extended API Options)

Optionally output additional ZCL messages that would normally be masked by the XBee application.

Use this when debugging FOTA updates by enabling client-side messages to be sent out of the serial port.

The bits in this parameter are used to enable different kinds of normally-suppressed output:

Parameter range

0x00 - 0x0A (bitfield)

Unused bits must be set to **0**. These bits may be logically OR'ed together:

Bit	Description
0	Reserved
1	Output receive frames for FOTA update commands
2	Reserved
3	Output Extended Modem Status (0x98) frames instead of Modem Status (0x8A) frames when a Secure Session status change occurs

Default

0

UART interface commands

The following commands affect the UART serial interface.

BD (UART Baud Rate)

This command configures the serial interface baud rate for communication between the UART port of the device and the host.

The device interprets any value between 0x12C and 0x0EC400 as a custom baud rate. Custom baud rates are not guaranteed and the device attempts to find the closest achievable baud rate. After setting a non-standard baud rate, query **BD** to find the actual operating baud rate before applying changes.

Parameter range

Standard baud rates: 0x0 - 0x0A

Non-standard baud rates: 0x12C - 0x0EC400

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

Parameter	Description
0x7	115200 b/s
0x8	230,400 b/s
0x9	460,800 b/s
0xA	921,600 b/s

Default

3 (9600 baud)

NB (Parity)

Set or read the serial parity settings for UART communications.

The device does not actually calculate and check the parity. It only interfaces with devices at the configured parity and stop bit settings for serial error detection.

Parameter range

0 - 2

Parameter	Description
0	No parity
1	Even parity
2	Odd parity

Default

0

SB (Stop Bits)

Sets or displays the number of stop bits for UART communications.

Parameter range

0 - 1

Parameter	Description
0	One stop bit
1	Two stop bits

Default

0

FT (Flow Control Threshold)

Set or display the flow control threshold.

The device de-asserts $\overline{CTS}$ when **FT** bytes are in the UART receive buffer. It re-asserts $\overline{CTS}$ when somewhat less than **FT** bytes are in the UART receive buffer. "Somewhat less than" allows for hysteresis so that $\overline{CTS}$ is not toggling rapidly when close to **FT** bytes are in the UART receive buffer.

Parameter range

0x20 - 0x1B0 bytes

Default

0x158

RO (Packetization Timeout)

Set or read the number of character times of inter-character silence required before transmission begins when operating in Transparent mode. A “character time” is the amount of time it takes to send a single ASCII character at the operating baud rate (**BD**).

Set **RO** to 0 to transmit characters as they arrive instead of buffering them into one RF packet.

The **RO** command only applies to Transparent mode, it does not apply to API mode.

Parameter range

0 - 0xFF (x character times)

Default

3

AT Command options

The following commands affect how Command mode operates.

CC (Command Character)

Sets or displays the character value used to break from data mode to Command mode. The command character must be sent three times in succession while observing the minimum guard time (**GT**) of silence before and after this sequence.

The default value (**0x2B**) is the ASCII code for the plus (**+**) character. You must enter it three times within the guard time to enter Command mode. To enter Command mode, there is also a 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. For more information, see Enter Command mode.

Parameter range

0 - 0xFF

Recommended: 0x20 - 0x7F (ASCII)

Default

0x2B (the ASCII plus character: **+**)

CT (Command Mode Timeout)

Sets or displays the Command mode timeout parameter. If the local device enters Command mode and does not receive any valid AT commands within this time period, Command mode silently exits.

Parameter range

2 - 0x1770 (x 100 ms)

Default

0x64 (10 seconds)

GT (Guard Times)

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 if a data stream in Transparent mode includes the **CC** character. For more information, see Enter Command mode.

Parameter range

0x2 - 0x6D3 (x 1 ms)

Default

0x3E8 (one second)

CN (Exit Command mode)

Executable command. **CN** immediately exits Command mode and applies pending changes.

Parameter range

N/A

Default

N/A

UART pin configuration commands

The following commands are related to pin configuration for the UART interface.

D6 (DIO6/RTS Configuration)

Sets or displays the DIO6/$\overline{\text{RTS}}$ configuration (Micro pin 27/SMT pin 29/TH pin 16).

Parameter range

0, 1, 3 - 5

Parameter	Description
0	Disabled
1	$\overline{\text{RTS}}$ flow control
2	N/A
3	Digital input

Parameter	Description
4	Digital output, low
5	Digital output, high

Default

0

D7 (DIO7/CTS Configuration)

Sets or displays the DIO7/$\overline{\text{CTS}}$ configuration (Micro pin 24/SMT pin 25/TH pin 12).

Parameter range

0, 1, 3 - 7

Parameter	Description
0	Disabled
1	$\overline{\text{CTS}}$ flow control
2	N/A
3	Digital input
4	Digital output, low
5	Digital output, high
6	RS-485 enable, low Tx (0 V on transmit, high when idle)
7	RS-485 enable, high Tx (high on transmit, 0 V when idle)

Default

1

P3 (DIO13/UART_DOUT Configuration)

Sets or displays the DIO13/UART_DOUT configuration (Micro pin 3/SMT pin 3/TH pin 2).

Parameter range

0, 1, 3 - 5

Parameter	Description
0	Disabled
1	UART DOUT
2	N/A
3	Digital input

Parameter	Description
4	Digital output, low
5	Digital output, high

Default

1

P4 (DIO14/UART_DIN Configuration)

Sets or displays the DIO14/UART_DIN configuration (Micro pin 4/SMT pin 4/TH pin 3).

Parameter range

0, 1, 3 - 5

Parameter	Description
0	Disabled
1	UART DIN
2	N/A
3	Digital input
4	Digital output, low
5	Digital output, high

Default

1

SMT/MMT SPI interface commands

The following commands affect the SPI serial interface on SMT and MMT variants. These commands are not applicable to the through-hole variant of the XBee 3; see **D1** through **D4** and **P2** for through-hole SPI support.

P5 (DIO15/SPI_MISO Configuration)

Sets or displays the DIO15/SPI_MISO configuration (Micro pin 16/SMT pin 17). This only applies to surface-mount and micro devices.

Parameter range

0, 1, 4, 5

Parameter	Description
0	Disabled

Parameter	Description
1	SPI_MISO
2	N/A
3	N/A
4	Digital output, low
5	Digital output, high

Default

1

P6 (DIO16/SPI_MOSI Configuration)

Sets or displays the DIO16/SPI_MOSI configuration (Micro pin 15/SMT pin 16). This only applies to surface-mount and micro devices.

Parameter range

0, 1, 4, 5

Parameter	Description
0	Disabled
1	SPI_MOSI
2	N/A
3	N/A
4	Digital output, low
5	Digital output, high

Default

1

P7 (DIO17/SPI_SSEL Configuration)

Sets or displays the DIO17/SPI_$\overline{\text{SSEL}}$ configuration (Micro pin 14/SMT pin 15). This only applies to surface-mount and micro devices.

Parameter range

0, 1, 4, 5

Parameter	Description
0	Disabled

Parameter	Description
1	SPI_$\overline{\text{SSEL}}$
2	N/A
3	N/A
4	Digital output, low
5	Digital output, high

Default

1

P8 (DIO18/SPI_CLK Configuration)

Sets or displays the DIO18/SPI_CLK configuration (Micro pin 13/SMT pin 14). This only applies to surface-mount and micro devices.

Parameter range

0, 1, 4, 5

Parameter	Description
0	Disabled
1	SPI_CLK
2	N/A
3	N/A
4	Digital output, low
5	Digital output, high

Default

1

P9 (DIO19/SPI_ATTN Configuration)

Sets or displays the DIO19/SPI_$\overline{\text{ATTN}}$ configuration (Micro pin 11/SMT pin 12). This only applies to surface-mount and micro devices.

Parameter range

0, 1, 4, 5

Parameter	Description
0	Disabled

Parameter	Description
1	SPI_$\overline{\text{ATTN}}$
2	N/A
3	N/A
4	Digital output, low
5	Digital output, high

Default

1

I/O settings commands

The following commands configure the various I/O lines available on the XBee 3 802.15.4 RF Module.

D0 (DIO0/ADC0/Commissioning Configuration)

Sets or displays the DIO0/ADC0/CB configuration (TH pin 20/SMT 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

CB (Commissioning Button)

Use **CB** to simulate Commissioning Pushbutton presses in software.

You can enable a physical commissioning pushbutton with D0 (DIO0/ADC0/Commissioning Configuration).

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.

Parameter range

1, 4

Parameter	Description
1	Keeps device awake for 30 seconds.
4	Restore defaults (equivalent to sending an RE (Restore Defaults)).

Default

N/A

D1 (DIO1/ADC1/TH_SPI_ATTN Configuration)

Sets or displays the DIO1/ADC1/TH_SPI_ATTN configuration (Micro pin 30/SMT pin 32/TH pin 19).

Parameter range

SMT/MMT: 0, 2 - 5

TH: 0 - 5

Parameter	Description
0	Disabled
1	SPI_ATTN for the through-hole device N/A for surface-mount device
2	ADC
3	Digital input
4	Digital output, low
5	Digital output, high

Default

0

D2 (DIO2/ADC2/TH_SPI_CLK Configuration)

Sets or displays the DIO2/ADC2/TH_SPI_CLK configuration (Micro pin 29/SMT pin 31/TH pin 18).

Parameter range

SMT/MMT: 0, 2 - 5

TH: 0 - 5

Parameter	Description
0	Disabled
1	SPI_CLK for through-hole devices N/A for surface-mount devices
2	ADC

Parameter	Description
3	Digital input
4	Digital output, low
5	Digital output, high

Default

0

D3 (DIO3/ADC3/TH_SPI_SSEL Configuration)

Sets or displays the DIO3/ADC3/TH_SPI_$\overline{\text{SSEL}}$ configuration (Micro pin 28/SMT pin 30/TH pin 17).

Parameter range

SMT/MMT: 0, 2 - 5

TH: 0 - 5

Parameter	Description
0	Disabled
1	SPI_$\overline{\text{SSEL}}$ for the through-hole device N/A for surface-mount device
2	ADC
3	Digital input
4	Digital output, low
5	Digital output, high

Default

0

D4 (DIO4/TH_SPI_MOSI Configuration)

Sets or displays the DIO4/TH_SPI_MOSI configuration (Micro pin 23/SMT pin 24/TH pin 11).

Parameter range

SMT/MMT: 0, 3 - 5

TH: 0, 1, 3 - 5

Parameter	Description
0	Disabled
1	SPI_MOSI for the through-hole device N/A for the surface-mount and micro device

Parameter	Description
2	N/A
3	Digital input
4	Digital output, low
5	Digital output, high

Default

0

D5 (DIO5/Associate Configuration)

Sets or displays the DIO5/ASSOCIATED_INDICATOR configuration (Micro pin 26/SMT pin 28/TH pin 15).

Parameter range

0, 1, 3 - 5

Parameter	Description
0	Disabled
1	Associate LED indicator - blinks when associated
2	N/A
3	Digital input
4	Digital output, default low
5	Digital output, default high

Default

1

D8 (DIO8/DTR/SLP_Request Configuration)

Sets or displays the DIO8/$\overline{\text{DTR}}$/SLP_RQ configuration (Micro pin 9/SMT pin 10/TH pin 9).

Note If **D8** is configured as DTR/Sleep_Request (**1**), the line will be left floating while the device sleeps. Leaving **D8** set to **1** and the corresponding pin not connected to anything external to the device may result in higher sleep current draw.

Parameter range

0, 1, 3 - 5

Parameter	Description
0	Disabled

Parameter	Description
1	DTR/Sleep_Request (used with pin sleep and cyclic sleep with pin wake)
2	N/A
3	Digital input
4	Digital output, low
5	Digital output, high

Default

1

D9 (DIO9/ON_SLEEP Configuration)

Sets or displays the DIO9/ON_SLEEP configuration (Micro pin 25/SMT pin 26/TH pin 13).

Parameter range

0, 1, 3 - 5

Parameter	Description
0	Disabled
1	ON/SLEEP indicator
2	N/A
3	Digital input
4	Digital output, low
5	Digital output, high

Default

1

P0 (DIO10/RSSI/PWM0 Configuration)

Sets or displays the DIO10/RSSI/PWM0 configuration (Micro pin 7/SMT pin 7/TH pin 6).

When configured as RSSI PWM output, the device outputs a PWM signal with a duty cycle equivalent to the dBm of the received packet.

Use RP (RSSI PWM Timer) to configure the timeout.

When configured as PWM output (**2**): you can use **M0** to explicitly control the PWM0 output. When used with Analog line passing, PWM0 corresponds with ADC0.

Parameter range

0 - 5

Parameter	Description
0	Disabled
1	RSSI PWM output
2	PWM0 output. M0 (PWM0 Duty Cycle) or I/O line passing control the value.
3	Digital input
4	Digital output, low
5	Digital output, high

Default

1

P1 (DIO11/PWM1 Configuration)

Sets or displays the DIO11/PWM1 configuration (Micro pin 8/SMT pin 8/TH pin 7).

When configured as PWM output (**2**): you can use **M1** to explicitly control the PWM1 output. When used with Analog line passing, PWM corresponds with ADC1.

Parameter range

0, 2 - 5

Parameter	Description
0	Disabled
1	N/A
2	PWM1 output. M1 (PWM1 Duty Cycle) or I/O line passing control the value.
3	Digital input
4	Digital output, low
5	Digital output, high

Default

0

P2 (DIO12/TH_SPI_MISO Configuration)

Sets or displays the DIO12/TH_SPI_MISO configuration (Micro pin 5/SMT pin 5/TH pin 4).

Parameter range

SMT/MMT: 0, 3 - 5

TH: 0, 1, 3 - 5

Parameter	Description
0	Disabled
1	SPI_MISO for the through-hole device N/A for the surface-mount and micro device
2	N/A
3	Digital input
4	Digital output, low
5	Digital output, high

Default

0

PR (Pull-up/Down Resistor Enable)

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.

The **PD** (Pull Direction) parameter determines the direction of the internal pull-up/down resistor.
PR and **PD** only affect lines that are configured as digital inputs (**3**) or disabled (**0**).
By default, pull-up resistors are enabled on all disabled I/O lines.
The following table defines the bit-field map for **PR** and **PD** commands.

Bit	I/O line	Micro pin	Surface-mount pin	Through-hole pin
0	DIO4	23	24	11
1	DIO3	28	30	17
2	DIO2	29	31	18
3	DIO1	30	32	19
4	DIO0	31	33	20
5	DIO6	27	29	16
6	DIO8	9	10	9
7	DIO14	4	4	3
8	DIO5	26	28	15
9	DIO9	25	26	13
10	DIO12	5	5	4
11	DIO10	7	7	6

Bit	I/O line	Micro pin	Surface-mount pin	Through-hole pin
12	DIO11	8	8	7
13	DIO7	24	25	12
14	DIO13	3	3	2
15	DIO15	16	17	N/A
16	DIO16	15	16	N/A
17	DIO17	14	15	N/A
18	DIO18	13	14	N/A
19	DIO19	11	12	N/A

Parameter range

Through-hole: 0 - 0xFFFF

SMT/MMT: 0 - 0xFFFFF

Default

0xFFFF

Example

Sending the command **ATPR 6F** turn bits 0, 1, 2, 3, 5 and 6 ON, and bits 4 and 7 OFF. The binary equivalent of 0x6F is 01101111. Bit 0 is the right-most digit in the binary bit field.

PD (Pull Up/Down Direction)

See PR (Pull-up/Down Resistor Enable) for the bit mappings.

Parameter range

Through-hole: 0 - 0xFFFF

SMT/MMT: 0 - 0xFFFFF

Default

0xFFFF

M0 (PWM0 Duty Cycle)

The duty cycle of the PWM0 line (Micro pin 7/SMT pin 7).

If IA (I/O Input Address) is set correctly and P0 (DIO10/RSSI/PWM0 Configuration) is configured as PWM0 output, incoming AD0 samples automatically modify the PWM0 value. See PT (PWM Output Timeout).

To configure the duty cycle of PWM0:

1. Enable PWM0 output (**P0** = **2**).
2. Change **M0** to the desired value.
3. Apply settings (use **CN** or **AC**).

The PWM period is 64 µs and there are 0x03FF (1023 decimal) steps within this period. When **M0** = **0** (0% PWM), 0x01FF (50% PWM), 0x03FF (100% PWM), and so forth.

Parameter range

0 - 0x3FF

Default

0

M1 (PWM1 Duty Cycle)

If IA (I/O Input Address) is set correctly and P1 (DIO11/PWM1 Configuration) is configured as PWM1 output, incoming AD0 samples automatically modify the PWM1 value. See PT (PWM Output Timeout).

To configure the duty cycle of PWM1:

1. Enable PWM1 output (**P1** = 2).
2. Change **M1** to the desired value.
3. Apply settings (use **CN** or **AC**).

The PWM period is 64 µs and there are 0x03FF (1023 decimal) steps within this period. When **M1** = **0** (0% PWM), 0x01FF (50% PWM), 0x03FF (100% PWM), and so forth.

Parameter range

0 - 0x3FF

Default

0

RP (RSSI PWM Timer)

The PWM timer expiration in 0.1 seconds. **RP** sets the duration of pulse width modulation (PWM) signal output on the RSSI pin. The pin signal duty cycle updates with each received packet and shuts off when the timer expires. This command is only applicable when **P0** is set to **1** which enables RSSI PWM output.

When **RP** = **0xFF**, the output is always on.

Parameter range

0 - 0xFF (x 100 ms), 0xFF

Default

0x28 (four seconds)

LT (Associate LED Blink Time)

Set or read the Associate LED blink time. If you use D5 (DIO5/Associate Configuration) 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.

Parameter range

0, 0x14 - 0xFF (x 10 ms)

Default

0

I/O sampling commands

The following commands configure I/O sampling on an originating device. Any I/O sample generated by this device is sent to the address specified by **DH** and **DL**. You must configure at least one I/O line as an input or output for a sample to be generated.

IS (I/O Sample)

Immediately forces an I/O sample to be generated for the digital and analog I/O lines that are configured for the local device. If you issue the command to the local device, the sample data is sent out the local serial interface. If sent remotely, the sample is taken on the destination and the sample data is returned as an Description.

If the device receives ERROR as a response to an **IS** query, there are no valid I/O lines to sample.

The **IS** command cannot be issued from within MicroPython or over BLE.

Refer to On-demand sampling for more information on using this command and examples.

Standard I/O capability

If AO (API Output Options) is set to **2**, the XBee 3 802.15.4 RF Module's **IS** I/O options are D0 (DIO0/ADC0/Commissioning Configuration) - D8 (DIO8/DTR/SLP_Request Configuration) and four analog channels: AD0/DIO0 - AD3/DIO3.

When operating in Transparent mode (AP (API Enable) = **0** and AO (API Output Options) = **2**), the data is returned in the following format:

All bytes are converted to ASCII:

number of samples<CR>

AIO/DIO mask (Bits 0 - 8 are digital I/O; Bits 9 - 12 analog channels)<CR>

DIO data<CR> (If DIO lines are enabled)

ADC channel Data<CR> (This will repeat for every enabled ADC channel)

<CR> (end of data noted by extra <CR>)

When operating in API mode (**AP** = 1), the command immediately returns an **OK** response. The data follows in the normal API format for DIO data.

Extended I/O capability

If **A0** is set to **0** or **1**, the XBee 3 802.15.4 RF Module's **IS** I/O options are D0 (DIO0/ADC0/Commissioning Configuration) - D9 (DIO9/ON_SLEEP Configuration) and P0 (DIO10/RSSI/PWM0 Configuration) - P4 (DIO14/UART_DIN Configuration) and four analog channels AD0/DIO0 - AD3/DIO3.

When operating in Transparent mode (**AP** = **0** and **AO** = **0**, AO = 1), the data is returned in the following format:

All bytes are converted to ASCII:

number of samples<CR>

DIO mask (Bits 0 - 14 are digital I/O<CR>

AIO mask (Bits 0 - 3 are Analog channels<CR>

DIO data<CR> (If DIO lines are enabled)

ADC channel Data<CR> (This will repeat for every enabled ADC channel)

<CR> (end of data noted by extra <CR>)

When operating in API mode (**AP** = **1**), the command immediately returns an **OK** response. The data follows in the normal API format for DIO data.

Parameter range

N/A

Default

N/A

IR (Sample Rate)

Set or read the I/O sample rate to enable periodic sampling. When set, this parameter causes the device to sample all enabled DIO and ADC at a specified interval.

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 (see D0 (DIO0/ADC0/Commissioning Configuration)-D8 (DIO8/DTR/SLP_Request Configuration), P0 (DIO10/RSSI/PWM0 Configuration)-P2 (DIO12/TH_SPI_MISO Configuration).

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

IC (DIO Change Detect)

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** - **P5**). If the device detects a change on an enabled digital I/O pin, it immediately transmits a digital I/O sample to the address specified by **DH** + **DL**. If sleep is enabled, the edge transition must occur during a wake period to trigger a change detect.

The data transmission contains only DIO data.

IC is a bitmask you can use to enable or disable edge detection on individual digital I/O lines. Only DIO0 through DIO15 can be sampled using a Change Detect.

Bit field

Bit	I/O line	Device pin
0	DIO0	Micro pin 31/SMT pin 33/TH pin 20
1	DIO1	Micro pin 30/SMT pin 32/TH pin 19

Bit	I/O line	Device pin
2	DIO2	Micro pin 29/SMT pin 31/TH pin 18
3	DIO3	Micro pin 28/SMT pin 30/TH pin 17
4	DIO4	Micro pin 23/SMT pin 24/TH pin 11
5	DIO5	Micro pin 26/SMT pin 28/TH pin 15
6	DIO6	Micro pin 27/SMT pin 29/TH pin 16
7	DIO7	Micro pin 24/SMT pin 25/TH pin 12
8	DIO8	Micro pin 9/SMT pin 10/TH pin 9
9	DIO9	Micro pin 25/SMT pin 26/TH pin 13
10	DIO10	Micro pin 7/SMT pin 7/TH pin 6
11	DIO11	Micro pin 8/SMT pin 8/TH pin 7
12	DIO12	Micro pin 5/SMT pin 5/TH pin 4
13	DIO13	Micro pin 3/SMT pin 3/TH pin 2
14	DIO14	Micro pin 4/SMT pin 4/TH pin 3

Parameter range

0 - 0x7FFF

Default

0

AV (Analog Voltage Reference)

The analog voltage reference used for A/D sampling.

Parameter range

0 - 2

Parameter	Description
0	1.25 V reference
1	2.5 V reference
2	VDD reference

Default

0

IT (Samples before TX)

Sets or displays the number of samples to collect before transmitting data. The maximum number of samples is dependent on the number of enabled I/O lines and the maximum payload available.

If **IT** is set to a number too big to fit in the maximum payload, it is reduced such that it will fit. A query of **IT** after setting it reports the actual number of samples in a packet.

Parameter range

0x1 - 0xFF

Default

1

IF (Sleep Sample Rate)

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 (Sample Rate). In addition, setting **IF** to zero allows I/O samples to occur before the device goes to sleep and occur thereafter every wake cycle specified by **IR**.

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.

For more information, see the following commands:

- D0 (DIO0/ADC0/Commissioning Configuration) through D9 (DIO9/ON_SLEEP Configuration)
- P0 (DIO10/RSSI/PWM0 Configuration) through P2 (DIO12/TH_SPI_MISO Configuration)

Parameter range

0 – 0xFF

Default

1

IO (Digital Output Level)

Sets digital output levels. This allows DIO lines setup as outputs to be changed through Command mode.

Parameter range

8-bit bit map; each bit represents the level of an I/O line set up as an output

Default

N/A

I/O line passing commands

The following AT commands allow I/O line passing to be enabled and configure the timeout that will be used for each I/O line. Line Passing requires the device to receive an I/O sample from the address specified by **IA** and have an I/O lines configured as outputs that corresponds to inputs in the received I/O sample.

IA (I/O Input Address)

The source address of the device to which outputs are bound.

To disable I/O line passing, set all bytes to **0xFF**.

To allow any I/O packet addressed to this device (including broadcasts) to change the outputs, set **IA** to **0xFFFF**.

Parameter range

0 - 0xFFFF FFFF FFFF FFFF

Default

0xFFFFFFFFFFFFFFFF (I/O line passing disabled)

IU (I/O Output Enable)

IU disables or enables I/O API UART output when line passing is enabled if the received sample has a source address that matches IA (I/O Input Address) or if **IA** is set to **0xFFFF**.

Note To enable API output, you must set AP (API Enable) to an API mode (**AP** = **1** or **2**).

Parameter range

0 - 1

Parameter	Description
0	Disabled
1	Enabled

Default

1

T0 (D0 Timeout Timer)

Specifies how long pin D0 (DIO0/ADC0/Commissioning Configuration) holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0xFF

Default

0

T1 (D1 Output Timeout Timer)

Specifies how long pin D1 (DIO1/ADC1/TH_SPI_ATTN Configuration) holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0xFF

Default

0

T2 (D2 Output Timeout Timer)

Specifies how long pin D2 (DIO2/ADC2/TH_SPI_CLK Configuration) holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0xFF

Default

0

T3 (D3 Output Timeout Timer)

Specifies how long pin D3 (DIO3/ADC3/TH_SPI_SSEL Configuration) holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0xFF

Default

0

T4 (D4 Output Timeout Timer)

Specifies how long pin D4 (DIO4/TH_SPI_MOSI Configuration) holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0xFF

Default

0

T5 (D5 Output Timeout Timer)

Specifies how long pin D5 (DIO5/Associate Configuration) holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0xFF

Default

0

T6 (D6 Output Timeout Timer)

Specifies how long pin D6 (DIO6/RTS Configuration) holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0xFF

Default

0

T7 (D7 Output Timeout Timer)

Specifies how long pin D7 (DIO7/CTS Configuration) holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0xFF

Default

0

T8 (D8 Output Timer)

Specifies how long pin D8 (DIO8/DTR/SLP_Request Configuration) holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0xFF

Default

0

T9 (D9 Output Timer)

Specifies how long pin D9 (DIO9/ON_SLEEP Configuration) holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0xFF

Default

0

Q0 (P0 Output Timer)

Specifies how long pin **P0** holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0xFF

Default

0

Q1 (P1 Output Timer)

Specifies how long pin P1 holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0xFF

Default

0

Q2 (P2 Output Timer)

Specifies how long pin P2 holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0xFF

Default

0

PT (PWM Output Timeout)

Specifies how long both PWM outputs (**P0**, **P1**) output a given PWM signal before it reverts to the configured value (**M0**/**M1**). If set to **0**, there is no timeout. This timeout only affects these pins when they are configured as PWM output.

Parameter range

0 - 0xFF (x 100 ms)

Default

0xFF

Location commands

The following commands are user-defined parameters used to store the physical location of the deployed device.

LX (Location X—Latitude)

User-defined GPS latitude coordinates of the node that is displayed on Digi Remote Manager and Network Assistant.

Parameter range

0 - 15 ASCII characters

Default

One ASCII space character (0x20)

LY (Location Y—Longitude)

User-defined GPS longitude coordinates of the node that is displayed on Digi Remote Manager and Network Assistant.

Parameter range

0 - 15 ASCII characters

Default

One ASCII space character (0x20)

LZ (Location Z—Elevation)

User-defined GPS elevation of the node that is displayed on Digi Remote Manager and Network Assistant.

Parameter range

0 - 15 ASCII characters

Default

One ASCII space character (0x20)

Diagnostic commands - firmware/hardware information

The following read-only commands are diagnostics that provide more information about the device.

VR (Firmware Version)

Reads the firmware version on a device.

Parameter range

0x2000 - 0x2FFF

Default

Set in the firmware

VL (Version Long)

Shows detailed version information including the application build date and time.

Parameter range

N/A

Default

N/A

VH (Bootloader Version)

Reads the bootloader version of the device.

Parameter range

N/A

Default

N/A

HV (Hardware Version)

Display the hardware version number of the device.

Parameter range

0 - 0xFFFF [read-only]

Pre-defined **HV** values for XBee 3 RF devices:

- 0x41 = XBee 3 Micro (MMT) and Surface Mount (SMT)
- 0x42 = XBee 3 Through Hole (TH)

Default

Set in the factory

R? (Power Variant)

Specifies whether the device is a PRO or Non-PRO variant.

- 0 = PRO (+19 dBm output power)
- 1 = Non-PRO (+8 dBm output power)

Parameter range

0, 1 [read-only]

Default

N/A

%C (Hardware/Software Compatibility)

Specifies what firmware is compatible with this device's hardware. **%C** is compared to the to the "compatibility_number" field of the firmware configuration xml file. Firmware with a compatibility number lower than the value returned by **%C** cannot be loaded onto the board. If an invalid firmware is loaded, the device will not boot until a valid firmware is reloaded.

Parameter range

[read-only]

Default

N/A

%V (Supply Voltage)

Reads the voltage on the Vcc pin in mV.

Parameter range

0 - 0xFFFF (in mV) [read only]

Default

N/A

TP (Module Temperature)

The current module temperature in degrees Celsius. The temperature is represented in two's complement, as shown in the following example:

1 ℃ = 0x0001 and -1℃ = 0xFFFF

Parameter range

0 - 0xFFFF (Celsius) [read-only]

Default

N/A

CK (Configuration CRC)

Reads the cyclic redundancy check (CRC) of the current AT command configuration settings to determine if the configuration has changed.

After a firmware update this command may return a different value.

Parameter range

0 - 0xFFFF [read-only]

Default

N/A

%P (Invoke Bootloader)

Forces the device to reset into the bootloader menu.

This command can only be issued locally.

Parameter range

N/A

Default

N/A

Memory access commands

This section details the executable commands that provide memory access to the device.

FR (Software Reset)

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

AC (Apply Changes)

This command applies changes to all command parameters configured in Command mode.

Any of the following also applies changes the same as issuing an **AC** command:

- Exiting Command mode with a **CN** command.
- Exiting Command mode via timeout.
- Receiving a 0x08 API command frame.
- Issuing a 0x08 Local AT Command API frame.
- Issuing a remote 0x17 AT Command API frame with option bit 1 set.

Example: Altering the UART baud rate with the **BD** command does not change the operating baud rate until after an **AC** command is received; at this point, the interface immediately changes baud rates.

Parameter range

N/A

Default

N/A

WR (Write)

Immediately writes parameter values to non-volatile flash memory so they persist through a power cycle. Operating network parameters are persistent and do not require a **WR** command for the device to reattach to the network.

Note Once you issue a **WR** command, do not send any additional characters to the device until after you receive the **OK** response. Use the **WR** command sparingly; the device's flash supports a limited number of write cycles.

Parameter range

N/A

Default

N/A

RE (Restore Defaults)

Restore device parameters to factory defaults.

Does not exit out of Command mode.

Parameter range

N/A

Default

N/A

Custom Default commands

The following commands are used to assign custom defaults to the device. Send RE (Restore Defaults) to restore custom defaults. You must send these commands as local AT commands, they cannot be set using Remote AT Command Request - 0x17.

%F (Set Custom Default)

When **%F** is received, the XBee 3 802.15.4 RF Module takes the next command received and applies it to both the current configuration and the custom defaults, so that when defaults are restored with RE (Restore Defaults) the custom value is used.

Parameter range

N/A

Default

N/A

!C (Clear Custom Defaults)

Clears all custom defaults. This command does not change the current settings, but only changes the defaults so that RE (Restore Defaults) restores settings to the factory values.

Parameter range

N/A

Default

N/A

R1 (Restore Factory Defaults)

Restores factory defaults, ignoring any custom defaults set using %F (Set Custom Default).

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 the firmware, so we recommend building the ability to filter out additional API frames with unknown frame types into your software interface.

Use the AP command to set the operation mode

Use AP (API Enable) to specify the operation mode:

AP command setting	Description
AP = 0	Transparent operating mode, UART serial line replacement with API modes disabled. This is the default option.
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 frame format

An API frame consists of the following:

- Start delimeter
- Length
- Frame data
- Checksum

API operation (AP parameter = 1)

This is the recommended API mode for most applications. The following table shows the data frame structure when you enable this mode:

Frame fields	Byte	Description
Start delimiter	1	0x7E
Length	2 - 3	Most Significant Byte, Least Significant Byte
Frame data	4 - number (n)	API-specific structure
Checksum	n + 1	1 byte

Any data received prior to the start delimiter is silently discarded. If the frame is not received correctly or if the checksum fails, the XBee replies with a radio status frame indicating the nature of the failure.

API operation with escaped characters (AP parameter = 2)

Setting API to 2 allows escaped control characters in the API frame. Due to its increased complexity, we only recommend this API mode in specific circumstances. API 2 may help improve reliability if the serial interface to the device is unstable or malformed frames are frequently being generated.

When operating in API 2, if an unescaped 0x7E byte is observed, it is treated as the start of a new API frame and all data received prior to this delimiter is silently discarded. For more information on using this API mode, see the Escaped Characters and API Mode 2 in the Digi Knowledge base.

API escaped operating mode 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). The following table shows the structure of an API frame with escaped characters:

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	

Start delimiter field

This field indicates the beginning of a frame. It is always 0x7E. This allows the device to easily detect a new incoming frame.

Escaped characters in API frames

If operating in API mode with escaped characters (**AP** parameter = 2), when sending or receiving a serial 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 (XORed with 0x20).

The following data bytes need to be escaped:

- 0x7E: start delimiter
- 0x7D: escape character
- 0x11: XON
- 0x13: XOFF

To escape a character:

1. Insert 0x7D (escape character).
2. Append it with the byte you want to escape, XORed with 0x20.

In API mode with escaped characters, the length field does not include any escape characters in the frame and the firmware calculates the checksum with non-escaped data.

Example: escape an API frame

To express the following API non-escaped frame in API operating mode with escaped characters:

Start delimiter	Length	Frame type	Frame Data	Checksum
			Data	
7E	00 0F	17	01 00 13 A2 00 40 AD 14 2E FF FE 02 4E 49	6D

You must escape the 0x13 byte:

1. Insert a 0x7D.
2. XOR byte 0x13 with 0x20: 13 ⊕ 20 = 33

The following figure shows the resulting frame. Note that the length and checksum are the same as the non-escaped frame.

Start delimiter	Length	Frame type	Frame Data	Checksum
			Data	
7E	00 0F	17	01 00 7D 33 A2 00 40 AD 14 2E FF FE 02 4E 49	6D

The length field has a two-byte value that specifies the number of bytes in the frame data field. It does not include the checksum field.

Length field

The length field is a two-byte value that specifies the number of bytes contained in the frame data field. It does not include the checksum field.

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.

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
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 3 802.15.4 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

Frame descriptions

The following sections describe the API frames.

64-bit Transmit Request - 0x00

Response frame: Transmit Status - 0x89

Description

This frame type is used to send serial payload data as an RF packet to a remote device with a corresponding 64-bit IEEE address.

Note This frame format is deprecated and should only be used by customers who require compatibility with legacy Digi RF products. For new designs, we encourage you to use Transmit Request - 0x10 to initiate API transmissions.

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**	64-bit Transmit Request - **0x00**
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	**Destination address**	Set to the 64-bit IEEE address of the destination device. If set to **0x000000000000FFFF**, the broadcast address is used.
13	8-bit	**Options**	A bit field of options that affect the outgoing transmission: ■ **Bit 0**: Disable MAC ACK [**0x01**] ■ Bit 1: Reserved (set to 0) ■ **Bit 2**: Send packet with Broadcast PAN ID [**0x04**] • 802.15.4 firmwares only **Note** Option values may be combined. Set all unused bits to 0.
14-n	variable	**RF data**	The serial data to be sent to the destination. Use **NP** to query the maximum payload size that can be supported based on current settings.
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

Sending a unicast transmission to a device with the 64-bit address of **0013A20012345678** with the serial data "**TxData**".

The corresponding Transmit Status - 0x89 response with a matching Frame ID will indicate whether the transmission succeeded.

```
7E 00 11 00 52 00 13 A2 00 12 34 56 78 00 54 78 44 61 74 61 9E
```

Frame type	Frame ID	64-bit dest address	Tx options	RF data
0x00	0x52	0x0013A200 12345678	0x00	0x547844617461
Input	*Matches response*			*"TxData"*

64-bit broadcast

Sending a broadcast transmission of the serial data "**Broadcast**" and suppressing the corresponding response by setting Frame ID to **0**.

```
7E 00 14 00 00 00 00 00 00 00 00 FF FF 00 42 72 6F 61 64 63 61 73 74 6E
```

Frame type	Frame ID	64-bit dest address	Tx options	RF data
0x00	0x00	0x00000000 0000FFFF	0x00	0x42726F616463617374
Input	*Suppress response*	*Broadcast address*		*"Broadcast"*

16-bit Transmit Request - 0x01

Response frame: Transmit Status - 0x89

Description

This frame type is used to send serial payload data as an RF packet to a remote device with a corresponding 16-bit network address.

Note This frame format is deprecated and should only be used by customers who require compatibility with legacy Digi RF products. For new designs, we encourage you to use Transmit Request - 0x10 to initiate API transmissions.

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**	16-bit Transmit Request - **0x01**
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	**Destination address**	Set to the 16-bit network address of the destination device. If set to **0xFFFF**, the broadcast address is used.
7	8-bit	**Options**	A bit field of options that affect the outgoing transmission: ■ **Bit 0**: Disable MAC ACK [**0x01**] ■ Bit 1: Reserved (set to 0) ■ **Bit 2**: Send packet with Broadcast PAN ID [**0x04**] • 802.15.4 firmwares only **Note** Option values may be combined. Set all unused bits to 0.
8-n	variable	**RF data**	The serial data to be sent to the destination. Use **NP** to query the maximum payload size that can be supported based on current settings.
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.

16-bit unicast

Sending a unicast transmission to a device with the 16-bit address of **1234** with the serial data "**TxData**".

The corresponding Transmit Status - 0x89 response with a matching Frame ID will indicate whether the transmission succeeded.

```
7E 00 0B 01 87 12 34 00 54 78 44 61 74 61 EB
```

Frame type	Frame ID	16-bit dest address	Tx options	RF data
0x01	0x87	0x1234	0x00	0x547844617461
Input	*Matches response*			*"TxData"*

16-bit broadcast

Sending a broadcast transmission of the serial data "**Broadcast**" and suppressing the corresponding response by setting Frame ID to **0**.

```
7E 00 0E 01 00 FF FF 00 42 72 6F 61 64 63 61 73 74 6D
```

Frame type	Frame ID	16-bit dest address	Tx options	RF data
0x01	0x00	0xFFFF	0x00	0x42726F616463617374
Input	*Suppress response*	*Broadcast address*		*"Broadcast"*

Local AT Command Request - 0x08

Response frame: Description

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 Description 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 Description 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 Description 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: Description

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 Description 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 Description 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 0x88 - Local AT Command Response 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.

See Maximum payload for additional information on payload size restrictions.

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
- If transmitting to a 64-bit destination, set the 16-bit address field to**0xFFFE**

16-bit addressing

- For unicast transmissions, set the 16-bit address field to the address of the desired destination node
- To use 16-bit addressing, set the 64-bit address field to**0xFFFFFFFFFFFFFFFF**

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 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**. Zigbee coordinator address is **0x0000000000000000**. When using 16-bit addressing, set this field to **0xFFFFFFFFFFFFFFFF**.
13	16-bit	**16-bit destination address**	Set to the 16-bit network address of the destination device, if known. If transmitting to a 64-bit address, sending a broadcast, or the 16-bit address is unknown, set this field 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.
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**.

802.15.4

Bit	Meaning	Description
0	Disable ACK [**0x01**]	Disable acknowledgments on all unicasts.
1	Broadcast PAN [**0x02**]	Transmission is sent to all PANs.
2	Reserved	<set this bit to 0>
3	Reserved	<set this bit to 0>
4	Secure Session Encryption [**0x10**]	Encrypt payload for transmission across a Secure Session. Reduces maximum payload size by 4 bytes.

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	16-bit dest	Bcast radius	Options	RF data
0x10	0x52	0x0013A20012345678	0xFFFE	0x00	0x00	0x547844617461
Request	*Matches response*	*Destination*	*Unknown*	*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	16-bit dest	Bcast radius	Tx Options	RF data
0x10	0x00	0x000000000000FFFF	0xFFFE	0x01	0x00	0x42726F616463617374

Frame type	Frame ID	64-bit dest	16-bit dest	Bcast radius	Tx Options	RF data
Request	*Suppress response*	*Broadcast address*	*Reserved*	*Single hop broadcast*	*Will use **TO***	*"Broadcast"*

16-bit unicast

Sending a unicast transmission to a device with the 16-bit address of **1234** with the serial data "**TxData**". Disable retries and acknowledgments to prioritize performance over reliability. The corresponding Transmit Status - 0x89 response with a matching Frame ID can be used to verify that the transmission was sent.

```
7E 00 14 10 8D FF FF FF FF FF FF FF FF 12 34 00 01 54 78 44 61 74 61 DD
```

Frame type	Frame ID	64-bit dest	16-bit dest	Bcast radius	Tx Options	RF data
0x10	0x8D	0xFFFFFFFF FFFFFFFF	0x1234	0x00	0x01	0x547844617461
Request	*Matches response*	*Use 16-bit addressing*	*Destination*	N/A	*Disable retries*	*"TxData"*

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, serial data transmissions, ZDO command execution, third-party Zigbee interfacing, and advanced Zigbee operations.

Query NP (Maximum Packet Payload Bytes) to read the maximum number of payload bytes that can be sent.

See Maximum payload for additional information on payload size restrictions.

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
- If transmitting to a 64-bit destination, set the 16-bit address field to **0xFFFE**

16-bit addressing

- DigiMesh does not support 16-bit addressing. The 16-bit address field is considered reserved and should be set to **0xFFFE**
- For unicast transmissions, set the 16-bit address field to the address of the desired destination node
- To use 16-bit addressing, set the 64-bit address field to **0xFFFFFFFFFFFFFFFF**

Reserved endpoints

For serial data transmissions, the **0xE8** endpoint should be used for both source and destination endpoints.

Endpoints **0xDC** - **0xEE** are reserved for special use by Digi and should not be used in an application outside of the listed purpose. The XBee 802.15.4 firmware only supports digi-specific endpoints, endpoints used outside of this range will be interpreted as the **0xE8** data endpoint.

The active Digi endpoints are:

- **0xE8** - Digi Data endpoint
- **0xE6** - Digi Device Object (DDO) endpoint
- **0xE5** - XBee3 - Secure Session Server endpoint
- **0xE4** - XBee3 - Secure Session Client endpoint
- **0xE3** - XBee3 - Secure Session SRP authentication 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.

The following profile IDs are handled by the XBee natively, all others—such as Smart Energy and Home Automation—can be passed through to a host:

- **0xC105** - Digi profile ID
- **0x0000** - Zigbee device profile ID (ZDP)

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 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**. Zigbee coordinator address is **0x0000000000000000**. When using 16-bit addressing, set this field to **0xFFFFFFFFFFFFFFFF**.
13	16-bit	**16-bit destination address**	Set to the 16-bit network address of the destination device if known. If transmitting to a 64-bit address, sending a broadcast, or the 16-bit address is unknown, set this field 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. For ZDO and ZCL commands, the command frame is inserted here. The fields in this nested command frame are represented in little-endian.
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**.

802.15.4

Bit	Meaning	Description
0	Disable ACK [**0x01**]	Disable acknowledgments on all unicasts.
1	Broadcast PAN [**0x02**]	Transmission is sent to all PANs.
2	Reserved	<set this bit to 0>
3	Reserved	<set this bit to 0>
4	Secure Session Encryption [**0x10**]	Encrypt payload for transmission across a Secure Session. Reduces maximum payload size by 4 bytes.

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	16-bit dest	Source EP	Dest EP	Cluster	Profile	Bcast radius	Tx options	Command data
0x11	0x87	0x0013A200 12345678	0xFFFE	0xE8	0xE8	0x0011	0xC105	0x00	0x00	0x5478446174 61
Explicit request	*Matches response*	*Destination*	*Unknown*	*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	16-bit dest	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*	*Unknown*	*Digi data*	*Digi data*	*Data*	*Digi profile*	*N/A*	*Disable retries*	*"TxData"*

ZDO command - ZDP Management Leave Request

Request a Zigbee device with the 64-bit address of **0013A20012345678** leave the network via a ZDO command. The ZDP request is sent as a broadcast with the destination defined in the ZDO command frame. Each field in the ZDO frame is in little-endian, the rest of the Digi API frame is big-endian.

In order to output the response to the ZDO command request, the sender must be configured to emit explicit receive frames by setting bit 0 of AO (API Output Options)—**AO** = 1. See Receiving ZDO command and responses for more information.

The corresponding Extended Transmit Status - 0x8B response with a matching Frame ID will indicate whether the transmission succeeded. The destination will handle the request and return a response to the sender, which will be emitted as a Explicit Receive Indicator - 0x91 if enabled.

```
7E 00 1E 11 01 00 00 00 00 00 00 FF FF FF FE 00 00 00 34 00 00 00 00A1 78 56 34
21 00 A2 13 00 00 45
```

Frame type	Frame ID	64-bit dest	16-bit dest	Source EP	Dest EP	Cluster	Profile	Bcast radius	Tx options	Command data
11	DE	0000000000000FFFF	FFFE	00	00	0000	0000	00	00	*A1 78563421 00A21300 00*

Frame type	Frame ID	64-bit dest	16-bit dest	Source EP	Dest EP	Cluster	Profile	Bcast radius	Tx options	Command data
0x11	0xDE	0x0000000000000FFFF	0xFFFE	0x00	0x00	0x0034	0x0000	0x00	0x00	■ 0xA1 ■ 0x0013A20012345678 ■ 0x00
Explicit request	*Matches response*	*ZDO commands should be broadcasted*	*Reserved*	ZDO	ZDO	*Management Leave Request Cluster*	*Zigbee Device Profile (ZDP)*	*Use **BH***	*Use **TO***	■ *Sequence num* ■ *64-bit dest* ■ *Options*

Remote AT Command Request - 0x17

Response frame: 0x97 - Remote AT Command Response

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.

XBee 3 firmwares support secured remote configuration through a Secure Session. Refer to Secured remote AT commands for information on how to secure your devices against unauthorized remote configuration.

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. When using 16-bit addressing, set this field to **0xFFFFFFFFFFFFFFFF**.
13	16-bit	**16-bit destination address**	Set to the 16-bit network address of the destination device if known. If transmitting to a 64-bit address or the 16-bit address is unknown, set this field to **0xFFFE**.

Offset	Size	Frame Field	Description
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 ■ Bit 2: Reserved (set to 0) ■ Bit 3: Reserved (set to 0) ■ **Bit 4**: Send the remote command securely [**0x10**] • Requires a secure session be established with the destination **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	16-bit dest	Command options	AT command	Parameter value
0x17	0x27	0x0013A20012345678	0xFFFE	0x02	0x4E49	0x52656D6F7465
Request	*Matches response*		*Unknown*	*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	16-bit dest	Command options	AT command	Parameter value
0x17	0x68	0x0013A200 12345678	0xFFFE	0x00	0x4944	0x0451
Request	*Matches response*		*Unknown*	*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	16-bit dest	Command options	AT command	Parameter value
0x17	0xFA	0x0013A200 12345678	0xFFFE	0x00	0x5450	(omitted)
Request	*Matches response*		*Unknown*	*N/A*	*"TP"*	*Query the parameter*

BLE Unlock Request - 0x2C

Response frame: BLE Unlock Response - 0xAC

Description

This frame type is used to authenticate a connection on the Bluetooth interface and unlock the processing of AT command frames across this interface. The frame format for the BLE Unlock Request - 0x2C and BLE Unlock Response - 0xAC are identical.

The unlock process is an implementation of the SRP (Secure Remote Password) algorithm using the RFC5054 1024-bit group and the SHA-256 hash algorithm . The SRP identifying user name, commonly referred to as *I*, is fixed to the username **apiservice**.

Upon completion, each side will have derived a shared session key which is used to communicate in an encrypted fashion with the peer. Additionally, a Modem Status - 0x8A with the status code **0x32**

(Bluetooth Connected) is emitted. When an unlocked connection is terminated, a Modem Status frame with the status code **0x33 (Bluetooth Disconnected)** is emitted.

The following implementations are known to work with the BLE SRP implementation:

- github.com/cncfanatics/SRP

You need to modify the hashing algorithm to SAH256 and the values ofNandgto use the RFC5054 1024-bit group.

- github.com/cocagne/csrp
- github.com/cocagne/pysrp

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**	BLE Unlock Request - **0x2C** BLE Unlock Response - **0xAC**
4	8-bit	**Step**	Indicates the phase of authentication and interpretation of payload data: 1. Client presents *A* value 2. Server presents *B* and *salt* 3. Client present *M1* session key validation value 4. Server presents *M2* session key validation value and two 12-byte nonces See the phase tables below for more information. Step values greater than 0x80 indicate error conditions: **0x80** = Unable to offer *B*—cryptographic error with content, usually due to *A mod N == 0* **0x81** = Incorrect payload length **0x82** = Bad proof of key **0x83** = Resource allocation error **0x84** = Request contained a step not in the correct sequence
5-n	varies	**Payload**	Payload structure varies by Step value. Refer to the phase tables below for the structure of this field.
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte—between length and checksum.

Phase tables

The following fields are inserted as the payload data depending on the phase of the authentication process

Phase 1 (Client presents A)

Offset	Size	Frame Field	Description
5	1024-bit (128 bytes)	A	One-time ephemeral client public key. If the *A* value is zero, the server will terminate the connection.

Phase 2 (Server presentsBand salt)

Offset	Size	Frame Field	Description
5	32-bit (4 bytes)	Salt	The SRP Salt value from the **$S** command.
9	1024-bit (128 bytes)	B	One-time ephemeral host public key.

Phase 3 (Client presentsM1)

Offset	Size	Frame Field	Description
5	256-bit (32 bytes)	M1	SHA256 hash algorithm digest.

Phase 4 (Server presents M2)

Offset	Size	Frame Field	Description
5	256-bit (32 bytes)	M2	SHA256 hash algorithm digest .
37	96-bit (12 bytes)	Tx nonce	Random nonce used as the constant prefix of the counter block for encryption/decryption of data transmitted to the API service by the client.
49	96-bit (12 bytes)	Rx nonce	Random nonce used as the constant prefix of the counter block for encryption/decryption of data received by the client from the API service.

Upon completion of *M2* verification, the session key has been determined to be correct and the API service is unlocked and will allow additional API frames to be used. Content from this point will be encrypted using AES-256-CTR with the following parameters:

- **Key**: The entire 32-byte session key.
- **Counter**: 128 bits total, prefixed with the appropriate nonce shared during authentication. Initial remaining counter value is 1.

  The counter for data sent into the XBee API Service is prefixed with the TX *nonce* value—see the **Phase 4** table, above—and the counter for data sent by the XBee to the client is prefixed with the *RX nonce* value.

Examples

Example sequence to perform AT Command XBee API frames over BLE

1. Discover the XBee 3 802.15.4 RF Module through scanning for advertisements.
2. Create a connection to the GATT Server.
3. Optional, but recommended: request a larger MTU for the GATT connection.
4. Turn on indications for the API Response characteristic.
5. Perform unlock procedure using BLE Unlock Request - 0x2C unlock frames.
6. Once unlocked, you may send Local AT Command Request - 0x08 frames and receive AT Command Response frames received.
   a. For each frame to send, form the API Frame, and encrypt through the stream cipher as described in the unlock procedure.
   b. Write the frame using one or more write operations.
   c. When successful, the response arrives in one or more indications. If your stack does not do it for you, remember to acknowledge each indication as it is received. Note that you are expected to process these indications and the response data is not available if you attempt to perform a read operation to the characteristic.
   d. Decrypt the stream of content provided through the indications, using the stream cipher as described in the unlock procedure.

User Data Relay Input - 0x2D

Response frame: Transmit Status - 0x89

Output frame: Description

Description

This frame type is used to relay user data between local interfaces: MicroPython (internal interface), BLE, or the serial port. Data relayed to the serial port—while in API mode—will be output as a Description frame.

For information and examples on how to relay user data using MicroPython, see Send and receive User Data Relay frames in the MicroPython Programming Guide.

For information and examples on how to relay user data using BLE, see Communicate with a Micropython application in the XBee Mobile SDK user guide.

Use cases

- You can use this frame to send data to an external processor through the XBee UART/SPI via the BLE connection. Use a cellphone to send the frame with UART interface as a target. Data contained within the frame is sent out the UART contained within an Output Frame. The external processor then receives and acts on the frame.
- Use an external processor to output the frame over the UART with the BLE interface as a target. This outputs the data contained in the frame as the Output Frame over the active BLE connection via indication.
- An external processor outputs the Frame over the UART with the Micropython interface as a target. Micropython operates over the data and publishes the data to mqtt topic.

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**	User Data Relay Input - **0x2D**
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	8-bit	**Destination Interface**	The intended interface for the payload data: **0** = Serial port—SPI, or UART when in API mode **1** = BLE **2** = MicroPython
6-n	variable	**Data**	The user data to be relayed
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Error cases

Errors are reported in a Transmit Status - 0x89 frame that corresponds with the Frame ID of the Relay Data frame:

Error code	Error	Description
0x7C	Invalid Interface	The user specified a destination interface that does not exist or is unsupported.

Error code	Error	Description
0x7D	Interface not accepting frames	The destination interface is a valid interface, but is not in a state that can accept data. For example: UART not in API mode, BLE does not have a GATT client connected, or buffer queues are full.

If the message was relayed successfully, no status will be generated.

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.

Relay to MicroPython

A host device needs to pass the message "**Relay Data**" to a MicroPython application running on a local XBee device via the serial port.

A corresponding Transmit Status - 0x89 response with a matching Frame ID will indicate if there was a problem with relaying the data.

If successful, the XBee micropython application can call **relay.receive()** to retrieve the data.

```
7E 00 0D 2D 3D 02 52 65 6C 61 79 20 44 61 74 61 FC
```

Frame type	Frame ID	Destination interface	Data
0x2D	0x3D	0x02	0x52656C61792044617461
Input	*Matches response*	*MicroPython*	*"Relay Data"*

Secure Session Control - 0x2E

Response frame: 0xAE - Secure Session Response

Description

This frame type is used to control a secure session between a client and a server. If the remote node has a password set and you set the frame to login, this will establish a secure session that will allow secured messages to be passed between the server and client.

This frame is also used for clients to log out of an existing secure session.

Secure Sessions are end-to-end connections. If a login attempt is addressed to a broadcast address, the attempt will fail with an invalid value—status **0xA**—error.

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**	Secure Session Control - **0x2E**
4	64-bit	**64-bit destination address**	Set to the 64-bit IEEE address of the destination device. Set to a broadcast address (**0x000000000000FFFF**) to affect all active incoming sessions.
12	8-bit	**Secure Session options**	Bit field of options that alter the session behavior: ■ **Bit 0**: Client-side control: • [**0x00**] = **Login** - Log in to a server as a client. ◦ If this bit is clear, the local device will act as a client and initiate SRP authentication with the target server. • [**0x01**] = **Logou**t - Log out of an existing session as a client. ◦ If this bit is set, the local device will attempt to end an existing client-side session with the target server. ◦ When set, all other options, the timeout field, and password will be ignored. ■ **Bit 1**: Server-side control: • [**0x02**] = **Terminate Session** - If this bit is set, the server will end active incoming session(s). ◦ The address field can be set to a specific node or the broadcast address can be used to end all incoming sessions. ◦ Use Extended Modem Status - 0x98 frames to manage multiple incoming sessions. ■ **Bit 2**: Timeout type: • [**0x00**] = **Fixed timeout** - The session terminates after the timeout period has elapsed. • [**0x04**] = **Inter-packet timeout** - The timeout is refreshed every time a secure transmission occurs between client and server. **Note** Option values may be combined. Set all unused bits to 0.

Offset	Size	Frame Field	Description
13	16-bit	**Timeout**	Timeout value for the secure session in units of ⅒ th second. Accepts up to **0x4650** (30 minutes). A session with a timeout of **0x0000** is considered a yielding session. Yielding sessions will never time out, but if a server receives a request to start a session when it has the maximum incoming sessions, the oldest yielding session will be ended by the server to make room for the new session. Sessions with non-zero timeouts will never be ended in this way.
15-n	variable	**Password**	The password set on the remote node—up to 64 ASCII characters. Will be ignored if this frame is a logout or server termination frame.
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.

Secure Session Client - Login with fixed timeout

A change is needed to be made on a device that is secured against unauthorized configuration changes. A gateway that is authorized to make the change logs into the remote node for 5 minutes as a client using the following frame:

The corresponding Secure Session Response - 0xAE will indicate whether the login attempt succeeded.

```
7E 00 14 2E 00 13 A2 00 12 34 56 78 00 0B B8 50 41 53 53 57 4F 52 44 D2
```

Frame type	64-bit dest	Session options	Timeout	Password
0x2E	0x0013A200 12345678	0x00	0x02B8	0x50415353574F5244D2
Request		*Login Fixed*	*5 minutes*	*"PASSWORD"*

Secure Session Client - Login for streaming data

A large stream of data needs to be sent to a gateway that is secured against receiving unauthorized data. Because the data stream, and the gateway's ability to process the data is unknown, a Secure Session using a 60 second inter-packet timeout is established. The sending device logs into the gateway as a client using the following frame:

The corresponding Secure Session Response - 0xAE will indicate whether the login attempt succeeded.

```
7E 00 13 2E 00 00 00 00 00 00 00 00 04 02 58 52 6F 73 33 62 75 64 D1
```

Frame type	64-bit dest	Session options	Timeout	Password
0x2E	0x00000000 00000000	0x04	0x0258	0x526F7333627564
Request	*Zigbee coordinator*	*Login* *Inter-packet*	*60 seconds*	*"Ros3bud"*

64-bit Receive Packet - 0x80

Request frames:

- Transmit Request - 0x10
- Explicit Addressing Command Request - 0x11
- 64-bit Transmit Request - 0x00
- 16-bit Transmit Request - 0x01

Description

This frame type is emitted when a device configured with legacy API output—AO (API Output Options) = **2**—receives an RF data packet from a device configured to use 64-bit source addressing—**MY** = **0xFFFE**.

Note This frame format is deprecated and should only be used by customers who require compatibility with legacy Digi RF products. For new designs, we encourage you to use Receive Packet - 0x90 for reception of API transmissions.

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**	64-bit Receive Packet - **0x80**
4	64-bit	**64-bit source address**	The sender's 64-bit IEEE address.
12	8-bit	**RSSI**	Received Signal Strength Indicator. The Hexadecimal equivalent of (-dBm) value. For example if RX signal strength is -40 dBm, then 0x28 (40 decimal) is returned.

Offset	Size	Frame Field	Description
13	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**: 802.15.4 only - Packet was broadcast across all PANs [**0x04**] **Note** Option values may be combined.
14-n	variable	**RF 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** = **2**.

```
7E 00 11 80 00 13 A2 00 12 34 56 78 5E 01 54 78 44 61 74 61 11
```

Frame type	64-bit source	RSSI	Rx options	Received data
0x80	0x0013A200 87654321	0x5E	0x01	0x547844617461
Output		*-94 dBm*	*ACK was sent*	*"TxData"*

16-bit Receive Packet - 0x81

Request frames:

- Transmit Request - 0x10
- Explicit Addressing Command Request - 0x11
- 64-bit Transmit Request - 0x00
- 16-bit Transmit Request - 0x01

Description

This frame type is emitted when a device configured with legacy API output—AO (API Output Options) = **2**—receives an RF data packet from a device configured to use 16-bit source addressing—**MY** < **0xFFFE**.

Note This frame format is deprecated and should only be used by customers who require compatibility with legacy Digi RF products. For new designs, we encourage you to use Receive Packet - 0x90 for reception of API transmissions.

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**	16-bit Receive Packet - **0x81**
4	16-bit	**16-bit source address**	The sender's 16-bit network address.
6	8-bit	**RSSI**	Received Signal Strength Indicator. The Hexadecimal equivalent of (-dBm) value. For example if RX signal strength is -40 dBm, then 0x28 (40 decimal) is returned.

Offset	Size	Frame Field	Description
7	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**: 802.15.4 only - Packet was broadcast across all PANs [**0x04**] **Note** Option values may be combined.
8-n	variable	**RF 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 16-bit address of **1234** 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** = **2**.

```
7E 00 0B 81 12 34 5E 01 54 78 44 61 74 61 93
```

Frame type	64-bit source	RSSI	Rx options	Received data
0x80	0x1234	0x5E	0x01	0x547844617461
Output		*-94 dBm*	*ACK was sent*	*"TxData"*

64-bit I/O Sample Indicator - 0x82

Description

This frame type is emitted when a device configured with legacy API output—AO (API Output Options) = **2**— receives an I/O sample frame from a remote device configured to use 64-bit source addressing—**MY** = **0xFFFE**. Only devices running in API mode will send I/O samples out the serial port.

Note This frame format is deprecated and should only be used by customers who require compatibility with legacy Digi RF products. For new designs, we encourage you to use I/O Sample Indicator - 0x92 for reception of I/O samples.

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**	64-bit I/O Sample Indicator - **0x82**
4	64-bit	**64-bit source address**	The sender's 64-bit IEEE address.
12	8-bit	**RSSI**	Received Signal Strength Indicator. The Hexadecimal equivalent of (-dBm) value. For example if RX signal strength is -40 dBm, then 0x28 (40 decimal) is returned.
13	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**: 802.15.4 only - Packet was broadcast across all PANs [**0x04**] **Note** Option values may be combined.
14	8-bit	**Number of samples**	The number of sample sets included in the payload.

Offset	Size	Frame Field	Description
15	16-bit	**Sample mask**	Bit field that indicates which I/O lines on the remote are configured as inputs, 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**: ADC0 **bit 10**: ADC1 **bit 11**: ADC2 **bit 12**: ADC3 bit 13: N/A bit 14: N/A bit 15: N/A Each bit represents either a DIO line or ADC channel. Bit set to 1 if channel is active.
17	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 channel mask field.
19	16-bit variable	**Analog samples (if included)**	If the sample set includes any analog I/O lines, 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).

16-bit I/O Sample Indicator - 0x83

Description

This frame type is emitted when a device configured with legacy API output—AO (API Output Options) = **2**— receives an I/O sample frame from a remote device configured to use 64-bit source addressing—**MY** = **0xFFFE**. Only devices running in API mode will send I/O samples out the serial port.

Note This frame format is deprecated and should only be used by customers who require compatibility with legacy Digi RF products. For new designs, we encourage you to use I/O Sample Indicator - 0x92 for reception of I/O samples.

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**	16-bit I/O Sample Indicator - **0x83**
4	16-bit	**16-bit source address**	The sender's 16-bit network address.
6	8-bit	**RSSI**	Received Signal Strength Indicator. The Hexadecimal equivalent of (-dBm) value. For example if RX signal strength is -40 dBm, then 0x28 (40 decimal) is returned.
7	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**: 802.15.4 only - Packet was broadcast across all PANs [**0x04**] **Note** Option values may be combined.
8	8-bit	**Number of samples**	The number of sample sets included in the payload.

Offset	Size	Frame Field	Description
9	16-bit	**Sample mask**	Bit field that indicates which I/O lines on the remote are configured as inputs, 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**: ADC0 **bit 10**: ADC1 **bit 11**: ADC2 **bit 12**: ADC3 bit 13: N/A bit 14: N/A bit 15: N/A Each bit represents either a DIO line or ADC channel. Bit set to 1 if channel is active.
11	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 channel mask field.
13	16-bit variable	**Analog samples (if included)**	If the sample set includes any analog I/O lines, 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).

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.

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 Description 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 Description 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 °C*

Transmit Status - 0x89

Request frames:

- 64-bit Transmit Request - 0x00
- 16-bit Transmit Request - 0x01
- User Data Relay Input - 0x2D

Description

This frame type is emitted when a transmit request completes. The status field of this frame indicates whether the request succeeded or failed and the reason.

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 - **0x89**
4	8-bit	**Frame ID**	Identifies the data frame for the host to correlate with a prior request.

Offset	Size	Frame Field	Description
5	8-bit	**Delivery status**	Complete list of delivery statuses: **0x00** = Success **0x01** = No ACK received **0x02** = CCA failure **0x03** = Indirect message unrequested **0x04** = Transceiver was unable to complete the transmission **0x21** = Network ACK failure **0x22** = Not joined to network **0x2C** = Invalid frame values (check the phone number) **0x31** = Internal error **0x32** = Resource error - lack of free buffers, timers, etc. **0x34** = No Secure Session Connection **0x35** = Encryption Failure **0x74** = Message too long **0x76** = Socket closed unexpectedly **0x78** = Invalid UDP port **0x79** = Invalid TCP port **0x7A** = Invalid host address **0x7B** = Invalid data mode **0x7C** = Invalid interface. See User Data Relay Input - 0x2D. **0x7D** = Interface not accepting frames. See User Data Relay Input - 0x2D. **0x7E** = A modem update is in progress. Try again after the update is complete. **0x80** = Connection refused **0x81** = Socket connection lost **0x82** = No server **0x83** = Socket closed **0x84** = Unknown server **0x85** = Unknown error **0x86** = Invalid TLS configuration (missing file, and so forth) **0x87** = Socket not connected **0x88** = Socket not bound 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).

Delivery status codes

Protocol-specific status codes follow

XBee 3 802.15.4

0x00 = Success

0x01 = No ACK received

0x02 = CCA failure

0x03 = Indirect message unrequested

0x04 = Transceiver was unable to complete the transmission

0x21 = Network ACK failure

0x22 = Not joined to network
0x31 = Internal error
0x32 = Resource error - lack of free buffers, timers, etc.
0x34 = No Secure Session Connection
0x35 = Encryption Failure
0x74 = Message too long
0x7C = Invalid interface. See User Data Relay Input - 0x2D.
0x7D = Interface not accepting frames. See User Data Relay Input - 0x2D.

XBee Cellular

0x00 = Successful transmit
0x21 = Failure to transmit to cell network
0x22 = Not registered to cell network
0x2C = Invalid frame values (check the phone number)
0x31 = Internal error
0x32 = Resource error (retry operation later). See Socket limits in API mode for more information.
0x74 = Message too long
0x76 = Socket closed unexpectedly
0x78 = Invalid UDP port
0x79 = Invalid TCP port
0x7A = Invalid host address
0x7B = Invalid data mode
0x7C = Invalid interface. See User Data Relay Input - 0x2D.
0x7D = Interface not accepting frames. See User Data Relay Input - 0x2D.
0x7E = A modem update is in progress. Try again after the update is complete.
0x80 = Connection refused
0x81 = Socket connection lost
0x82 = No server
0x83 = Socket closed
0x84 = Unknown server
0x85 = Unknown error
0x86 = Invalid TLS configuration (missing file, and so forth)
0x87 = Socket not connected
0x88 = Socket not bound

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.

Successful transmission

Host sent a unicast transmission to a remote device using a 64-bit Transmit Request - 0x00 frame.

The corresponding 0x89 Transmit Status with a matching Frame ID is emitted as a response to the request:

```
7E 00 03 89 52 00 24
```

Frame type	Frame ID	Delivery status
0x89	0x52	0x00
Response	*Matches request*	*Success*

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**

Offset	Size	Frame Field	Description
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** = Remote Manager connected **0x0F** = 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 **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.

XBee 802.15.4

0x00 = Hardware reset or power up

0x01 = Watchdog timer reset

0x02 = End device successfully associated with a coordinator

0x03 = End device disassociated from coordinator or coordinator failed to form a new network

0x06 = Coordinator formed a new network

0x0D = Voltage supply limit exceeded—see **Over-voltage detection** in the *XBee 3 RF Module Hardware Reference Manual*.

0x3B = XBee 3 - Secure session successfully established
0x3C = XBee 3 - Secure session ended
0x3D = XBee 3 - Secure session authentication failed
0x32 = XBee 3 - BLE Connect
0x33 = XBee 3 - BLE Disconnect
0x34 = XBee 3 - No Secure Session Connection

XBee Cellular

0x00 = Hardware reset or power up
0x01 = Watchdog timer reset
0x02 = Registered with cellular network
0x03 = Unregistered with cellular network
0x0E = Remote Manager connected
0x0F = Remote Manager disconnected
0x32 = XBee 3 - BLE Connect
0x33 = XBee 3 - 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

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.

Zigbee transmissions to an unknown network address of **0xFFFE** will return the discovered 16-bit network address in this response frame. This network address should be used in subsequent transmissions to the specific destination.

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	**16-bit destination address**	The 16-bit network address where the packet was delivered (if successful). If not successful, this address is **0xFFFD** (destination address unknown). **0xFFFE** indicates 16-bit addressing was not used.
7	8-bit	**Transmit retry count**	The number of application transmission retries that occur.

Offset	Size	Frame Field	Description
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 **0x15** = Invalid destination endpoint **0x21** = Network ACK failure **0x22** = Not joined to network **0x23** = Self-addressed **0x24** = Address not found **0x25** = Route not found **0x26** = Broadcast source failed to hear a neighbor relay the message **0x2B** = Invalid binding table index **0x2C** = Resource error - lack of free buffers, timers, etc. **0x2D** = Attempted broadcast with APS transmission **0x2E** = Attempted unicast with APS transmission, but **EE** = **0** **0x31** = Internal resource error **0x32** = Resource error lack of free buffers, timers, etc. **0x34** = No Secure Session connection **0x35** = Encryption failure **0x74** = Data payload too large **0x75** = Indirect message unrequested Refer to the tables below for a filtered list of status codes that are appropriate for specific devices.
9	8-bit	**Discovery status**	Complete list of delivery statuses: **0x00** = No discovery overhead **0x01** = Zigbee address discovery **0x02** = Route discovery **0x03** = Zigbee address and route discovery **0x40** = Zigbee end device extended timeout
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Delivery status codes

Protocol-specific status codes follow

XBee 3 802.15.4

0x00 = Success

0x01 = MAC ACK Failure

0x02 = CCA failure

0x03 = Indirect message unrequested

0x21 = Network ACK Failure

0x31 = Internal resource error

0x34 = XBee 3 - No Secure Session Connection

0x35 = Encryption Failure

0x74 = Data payload too large

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.

Successful transmission

Host sent a unicast transmission to a remote Zigbee device using a Transmit Request - 0x10frame. The transmission was sent using the destination's IEEE 64-bit address with a 16-bit network address of 0xFFFE (unknown).

The corresponding Extended Transmit Status - 0x8B with a matching Frame ID is emitted as a response to the request:

```
7E 00 07 8B 52 12 34 02 00 01 D9
```

Frame type	Frame ID	16-bit dest address	Tx retries	Delivery status	Discovery status
0x8B	0x52	0x1234	0x02	0x00	0x01
Response	*Matches request*	*Discovered NWK address*	*2 retries*	*Success*	*Address discovery*

To reduce discovery overhead, the host can retrieve the discovered 16-bit network address from this response frame to use in subsequent transmissions.

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 Output 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	**16-bit source address**	The sender's 16-bit network address.

Offset	Size	Frame Field	Description
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**: 802.15.4 only - Packet was broadcast across all PANs [**0x04**] ■ **Bit 3**: Reserved ■ **Bit 4**: Packet was sent across a secure session [0x10] ■ **Bit 5**: Packet encrypted with Zigbee APS security [**0x20**] ■ **Bit 6**: Zigbee only - packet was sent from an End Device [**0x40**] ■ **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.
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 **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** = **0**.

```
7E 00 12 90 00 13 A2 00 87 65 43 21 56 14 01 54 78 44 61 74 61 B9
```

Frame type	64-bit source	16-bit source	Rx options	Received data
0x90	0x0013A200 87654321	0x5614	0x01	0x547844617461
Output		*Network address*	*ACK was sent*	*"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 Output 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.

This frame is also emitted as a response to ZDO command requests, see Receiving ZDO command and responsesfor more information. The Cluster ID and endpoints are used to identify the type of transaction that occurred.

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	**16-bit source address**	The sender's 16-bit network address.
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.

Offset	Size	Frame Field	Description
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**: 802.15.4 only - Packet was broadcast across all PANs [**0x04**] ■ **Bit 4**: Packet was sent across a secure session [**0x10**] ■ **Bit 5**: Packet encrypted with Zigbee APS security [**0x20**] ■ **Bit 6**: Zigbee only - packet was sent from an End Device [**0x40**] ■ **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 87 65 43 21 87 BD E8 E8 00 11 C1 05 01 54 78 44 61 74 61
37
```

Frame type	64-bit source	16-bit source	Source EP	Dest EP	Cluster	Profile	Rx options	Received data
0x91	0x0013A20087654321	0x87BD	0xE8	0xE8	0x0011	0xC105	0x01	0x547844617461
Explicit output		*Network address*	*Digi data*	*Digi data*	*Data*	*Digi profile*	*ACK was sent*	*"TxData"*

ZDO command - ZDP IEEE Address Response

A ZDP IEEE address request is issued in order to identify the 64-bit address of a Zigbee device with the 16-bit network address of 0x046D. The following response is emitted out of the device that issued the request if configured to do so. In order to output the response to the ZDO command request, the sender must be configured to emit explicit receive frames by setting bit 0 of AO (API Output Options) (**AO** = **1**). See Receiving ZDO command and responses for more information.

Note Each field in the ZDO frame is in little-endian, the rest of the Digi API frame is big-endian.

```
7E 00 1E 91 00 13 A2 00 12 34 56 78 04 6D 00 00 80 01 00 00 01 B5 00 78 56 34 12
00 A2 13 00 6D 04 C3
```

Frame type	64-bit source	16-bit source	Source EP	Dest EP	Cluster	Profile	Rx options	Received data
91	0013A200 12345678	046D	00	00	8001	0000	01	B5 00 78563412 00A21300 6D04
0x91	0x0013A200 87654321	0x046D	0x00	0x00	0x8001	0xC105	0x01	■ 0xB5 ■ 0x00 ■ 0x0013A200 12345678 ■ 0x046D
Explicit output		*Network address*	*ZDO*	*ZDO*	*IEEE Address Response*	*ZDO*	*ACK was sent*	■ *Sequence Num* ■ *Status* ■ *IEEE Address* ■ *NWK Address*

I/O Sample Indicator - 0x92

Description

This frame type is emitted when a device configured with standard API output—AO (API Output 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	**16-bit source address**	The sender's 16-bit network address.
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.

Offset	Size	Frame Field	Description
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 mask**	Bit field that indicates which I/O lines on the remote are configured as analog input, if any: **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 87 AC 01 01 00 38 06 00 28 02 25 00 F8 EA
```

Frame type	64-bit source	16-bit source	Rx options	Num samples	Digital channel mask	Analog channel mask	Digital samples	Analog sample 1	Analog sample 2
0x92	0x0013A200 12345678	0x87AC	0x01	0x01	0x0038	0x06	0x0028	0x0225	0x00F8
Sample		*Network address*	*ACK was sent*	*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; DI04 is LOW*	*AD1 data*	*AD2 data*

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	**16-bit source address**	The sender's 16-bit network address.
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 Statuses for Secured remote AT commands: **0x0B** = No Secure Session - Remote command access requires a secure session be established first **0x0C** = Encryption error **0x0D** = Command was sent insecurely - A Secure Session exists, but the request needs to have the appropriate command option set (bit 4).

Offset	Size	Frame Field	Description
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	16-bit source	AT command	Command Status	Command data
0x97	0x27	0x0013A200 12345678	0x127E	0x4E49	0x00	(omitted)
Response	*Matches request*		*Network address*	*"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	16-bit source	AT command	Command Status	Command data
0x97	0x27	0x0013A200 12345678	0xFFFE	0x4944	0x04	(omitted)
Response	*Matches request*		*Unknown*	*"ID"*	*Transmission failure*	*Parameter changes return no data*

Query remote command parameter

Query the temperature of a remote device—TP (Module 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	16-bit source	AT command	Command Status	Command data
0x97	0x27	0x0013A200 12345678	0x127E	0x4944	0x00	0x002F
Response	*Matches request*		*Network address*	*"TP"*	*Success*	*+47 °C*

Extended Modem Status - 0x98

Description

The Extended Modem Status - 0x98 frame is intended to provide additional in-frame diagnostic information over the traditional Modem Status - 0x8A 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**	Extended Modem Status - **0x98**
4	8-bit	**Status code**	Refer to the tables below for appropriate status codes
n	variable	**Status data (optional)**	Additional fields that provide information about the status
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Secure Session status codes

When AZ (Extended API Options) is configured to output extended secure session statuses, whenever Secure Session API Frames are emitted, the extended modem status will provide additional details about the event.

Status code	Description	Status data	Size	Description
0x3B	A Secure Session was established with this node	Address	64-bit	The address of the client in the session.
		Options	8-bit	Session options set by the client.
		Timeout	16-bit	Session timeout set by the client.
0x3C	A Secure Session ended	Address	64-bit	The address of the other node in this session.
		Reason	8-bit	The reason the session was ended: **0x00** - Session was terminated by the other node **0x01** - Session Timed out **0x02** - Received a transmission with an invalid encryption counter **0x03** - Encryption counter overflow - the maximum number of transmissions for a single session has been reached **0x04** - Remote node out of memory
0x3D	A Secure Session authentication attempt failed	Address	64-bit	Address of the client node.
		Error	8-bit	Error that caused the authentication to fail. See Secure Session Response - 0xAE for a list of error statuses.

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.

Secure Session established

A device has established a secure session with the local node that has AZ (Extended API Options) configured to output extended secure session information. The following frame is emitted that announces the secure session establishment.

```
7E 00 0D 98 3B 00 13 A2 00 12 34 56 78 00 46 50 CD
```

Frame type	Status code	Status data
0x98	0x3B	■ 0x0013A20012345678 ■ 0x00 ■ 0x4650

Frame type	Status code	Status data
Extended status	*Secure Session established*	■ Address ■ Options ■ Timeout (30 min)

Zigbee Verbose join messages

The following example shows a successful association of a device that has configured to enable Verbose Join messages. The device is operating in Transparent mode—**AP** = **0**—to allow a human-friendly way to troubleshoot association issues, if set for API mode—**AP** = **1**—equivalent 0x98 Extended Modem Status frames would be emitted.

Message	Description
V AI -SearchingforParent:FF	...search has started
V Scanning:03FFF800	...channels 11 through 25 are enabled by the SC setting for the Active Search.
V BeaconRsp:0000000000000042A6010B949AC8FF	■ **ZS** = 0x00 ■ extendedPanId = 00000000000042A6 ■ allowingJoin 0x01 (yes) ■ radiochannel 0x0B ■ panid 0x949A ■ rssi 0xC8 ■ lqi = 0xFF
V Reject ID	...beacon response's extendedPanId does not match this radio's ID setting of 3151
V BeaconRsp:0200000000000002AB010C55D2B2DB	■ **ZS** = 0x02 ■ extendedPanId = 0x00000000000002AB ■ allowingJoin = 0x01 (yes) ■ radiochannel = 0x0C ■ panid = 0x55D2 ■ rssi = 0xB2 ■ lqi = 0xDB
V Reject ZS	...beacon response's **ZS** does not match this radio's **ZS** setting
V BeaconRsp:00000000000000003151010EE29FDFFF	
V BeaconSaved:0E05E29F0000000000003151	...this beacon response is acceptable as a candidate for association
V Joining:0E05E29F0000000000003151	...sending association request

Message	Description
V StackStatus: joined, network up 0290	...we are joined, the network is up, we can send and transmit
V Joined unsecured network:	
V AI -AssociationSucceeded:00	

BLE Unlock Response - 0xAC

Request frame: BLE Unlock Request - 0x2C

Description

This frame type is emitted in response to a BLE Unlock Request - 0x2C during a multi-stage BLE authentication exchange.

This frame's format is identical to that of the originating request. Refer to BLE Unlock Request - 0x2C for information on the formatting and proper use of this frame.

User Data Relay Output - 0xAD

Input frame: User Data Relay Input - 0x2D

Description

This frame type is emitted when user data is relayed to the serial port from a local interface: MicroPython (internal interface), BLE, or the serial port.

For information and examples on how to relay user data using MicroPython, see Send and receive User Data Relay frames in the *MicroPython Programming Guide*.

for information and examples on how to relay user data using BLE, see Communicate with a Micropython application in the *XBee Mobile SDK user guide*.

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.

Offset	Size	Frame Field	Description
3	8-bit	**Frame type**	User Data Relay Output - **0xAD**
4	8-bit	**Source Interface**	The intended interface for the payload data: **0** = Serial port—SPI, or UART when in API mode **1** = BLE **2** = MicroPython
5-n	variable	**Data**	The user data to be relayed
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte (between length and checksum).

Error cases

Errors are reported in a Transmit Status - 0x89 frame that corresponds with the Frame ID of the Relay Data frame:

Error code	Error	Description
0x7C	Invalid Interface	The user specified a destination interface that does not exist or is unsupported.
0x7D	Interface not accepting frames	The destination interface is a valid interface, but is not in a state that can accept data. For example: UART not in API mode, BLE does not have a GATT client connected, or buffer queues are full.

If the message was relayed successfully, no status will be generated.

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.

Relay from Bluetooth (BLE)

A mobile phone sends a serial data message to the XBee device's BLE interface. The message is flagged to be sent out of the serial port of the XBee device. The following frame outputs the relayed data:

```
7E 00 0C AD 01 52 65 6C 61 79 20 44 61 74 61 BA
```

Frame type	Source interface	Data
0xAD	0x01	0x52656C61792044617461
Output	*Bluetooth*	*"Relay Data"*

Secure Session Response - 0xAE

Request frame: Secure Session Control - 0x2E

Description

This frame type is output as a response to a Secure Session Control - 0x2E attempt. It indicates whether the Secure Session operation was successful or not.

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**	Secure Session Response - **0xAE**
4	8-bit	Response type	The type of response to correlate with the preceding request: **0x00** - Login response **0x01** - Logout response **0x02** - Server Termination
5	64-bit	**64-bit source address**	The 64-bit IEEE address of the responding device.

Offset	Size	Frame Field	Description
13	8-bit	**Status**	Typical statuses: **0x00** - SRP operation was successful **0x01** - Invalid Password - SRP verification failed due to mismatched **M1** and **M2** values **0x02** - Session request was rejected as there are too many active sessions on the server already **0x03** - Session options or timeout are invalid **0x05** - Timed out waiting for the other node to respond **0x06** - Could not allocate memory needed for authentication **0x07** - A request to terminate a session in progress has been made **0x08** - There is no password set on the server **0x09** - There was no initial response from the server **0x0A** - Data within the frame is not valid or formatted incorrectly Atypical statuses: **0x80** - Server received a packet that was intended for a client or vice-versa **0x81** - Received an SRP packet we were not expecting **0x82** - Offset for a split value (A/B) came out of order **0x83** - Unrecognized or invalid SRP frame type **0x84** - Authentication protocol version is not supported **0xFF** - An undefined error occurred
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.

Secure Session Login attempt

A client attempted to log into a Secure Session server.

The following Secure Session Response - 0xAE is emitted as a response:

```
7E 00 0B AE 00 00 13 A2 00 12 34 56 78 00 88
```

Frame type	Response type	64-bit source	Status
0x2E	0x00	0x0013A200 12345678	0x00
Response	*Login*		*success*

OTA firmware/file system upgrades

Overview

The XBee 3 802.15.4 RF Module supports two kinds of over-the-air upgrades:

- Firmware upgrades: upgrading the firmware or bootloader code on a device remotely.
- File System upgrades: placing or replacing the entire file system on a remote device.

An OTA upgrade is performed using two XBee3 RF modules: The **client module** is the module being upgraded, and the **server module** is connected to an external processor (the **OTA upgrade server**) and used to send the upgrade to the client. XCTU and Network Manager are capable of acting as an OTA upgrade server, and are the recommended method for distributing OTA upgrades. See Create an OTA upgrade server for more information on the OTA upgrade protocol.

Firmware over-the-air upgrades

A firmware OTA upgrade upgrades either just the application firmware or both the application firmware and the bootloader firmware on a device. OTA firmware upgrades must be to a different version, re-installing the same version as what is already installed is not supported.

Note Performing an OTA upgrade will erase any file system or bundled MicroPython code on the target device, even if the OTA upgrade does not complete.

File system over-the-air upgrades

A file system OTA upgrade uses the same protocol as a firmware OTA upgrade, but instead of changing the device firmware it installs a new image to the target module's file system. This method does not allow writing individual files, only copying an entire file system image at once. See OTA file system upgrades for more information on creating and sending file system images.

Scheduled upgrades

When a client has finished downloading the data for an OTA upgrade, it sends a request to the server asking when to apply the upgrade. The server can instruct the client to upgrade immediately, to wait a specified amount of time before upgrading, or to wait for a further command from the server to upgrade. If instructed to wait, the device will keep the downloaded upgrade for the specified time and then apply it. If a client looses track of time—for example, due to power loss—it will attempt to re-send the request for an upgrade time to the server and resume waiting. If the device does not receive a response to this request after a number of attempts, it applies the upgrade immediately.

Note Sleeping devices do not count time towards the upgrade while asleep. The delay for a scheduled upgrade on a sleeping end device should be calculated only considering the time that device will be awake.

Different OTA upgrade server tools have varying levels of support for scheduled upgrades. See the documentation for the OTA upgrade server you are using, or see Create an OTA upgrade server for information on how to implement scheduled upgrades on a server.

Create an OTA upgrade server

ZCL firmware upgrade cluster specification

The process, format, and commands used for OTA firmware upgrades are based on the ZCL OTA Upgrade cluster from the ZCL specification. The specification used is in Zigbee document 07-5123-06. Chapter 2 describes the general format of ZCL commands and chapter 11 describes the OTA upgrade cluster in detail. The specification contains a complete description of the OTA upgrade process, and you should reference it when creating an OTA upgrade server. This guide focuses on differences and examples specific to the XBee 3 802.15.4 RF Module. Where relevant, we refer to the ZCL specification document by section, for example (ZCL Spec §11.2.1).

Differences from the ZCL specification

The OTA upgrade process differs from what is described in the ZCL specification in the following ways:

- Setting/querying OTA cluster attributes and parameters (ZCL Spec §11.10, §11.11) is not supported.
- The WAIT_FOR_DATA status in an Image Block Response Command (ZCL Spec §11.13.8) is not supported.
- Devices will not automatically discover an OTA upgrade server upon joining a network (ZCL Spec §11.8). To specify an OTA server set US (OTA Upgrade Server), or leave it at its default value to accept OTA upgrades from any server.
- Clients do not automatically query the server for an available upgrade. The only way to start an OTA upgrade is by sending an Image Notify command from the server.

OTA files

Use an OTA file to perform an OTA upgrade. The OTA file format consists of an OTA header describing what is present in the file followed by one or more sub-elements containing the upgrade data. The OTA file format is described in the ZCL Spec §11.4.

The OTA file is included alongside other firmware files in each release. The file with the .ota extension contains the application firmware update, and the file with the .otb extension contains updates for both the firmware and the bootloader. The recommended bootloader version is listed in each firmware release's XML file—if the target device has an older version, we strongly recommend that you perform the OTA update using the .otb file. Updating a device with the same or newer bootloader version as the recommended version will not change the bootloader, but will update the application.

OTA header

The OTA header contains information about the upgrade data contained in the file. An OTA server needs to parse this file in order to get information that will be requested by a file. The OTA header format is (ZCL Spec §11.4.2):

Offset	Length	Name	Description
0	4	OTA upgrade file identifier	Unique identifier for an OTA file - will always be 0x0BEEF11E.

Offset	Length	Name	Description
4	2	OTA header version	Version for the OTA header format - The OTA header version supported by XBee 3 firmwares is 0x0100.
6	2	OTA header length	The length in bytes of this OTA header.
8	2	OTA header field control	Indicates what optional fields are present.
10	2	Manufacturer code	The manufacturer code for the image.
12	2	Image type	One of two values: ■ 0x0000 for a firmware upgrade ■ 0x0100 for a file system upgrade
14	4	File version	Contains the version information for this upgrade. See File version definition for more information on how to interpret this field. **Note** It is important to parse this value from the OTA file itself instead of inferring it from the file name, as the software compatibility number is not included elsewhere.
18	2	Zigbee stack version	This field is not used for and can be ignored.
20	32	OTA header string	A human-readable string to identify the OTA file.
52	4	Total image size	The total size of the OTA file, including the OTA header. **Note** This field contains incorrect information in most older firmware files and should not be used in the update process. The total size of the file should be determined using an external method.

Note All fields—except for the OTA header string—are in little endian byte order. Optional fields may be present at the end of the OTA header, they have been omitted here as they are not used in the XBee 3 upgrade process.

File version definition

The file version is a 32-bit integer—sent in little-endian byte order—containing information on a firmware version. It is divided into two fields:

- The most significant byte corresponds to the compatibility number field in the firmware's XML file—see %C (Hardware/Software Compatibility)—for a description of the compatibility number's effect on loading firmware.
- The remaining three bytes indicate the firmware version as reported by **VR**.

For example, a file version of **0x0100100A** indicates that the software compatibility number is **1** and the version number is **100A**. **0x0200300B** indicates that the software compatibility number is **2** and the version is **300B**.

Sub-elements

All data after the OTA header is organized into sub-elements. Most OTA files will contain a single sub-element: the upgrade image. Sub-elements are arranged as tag-length-value triplets, as shown in the table below.

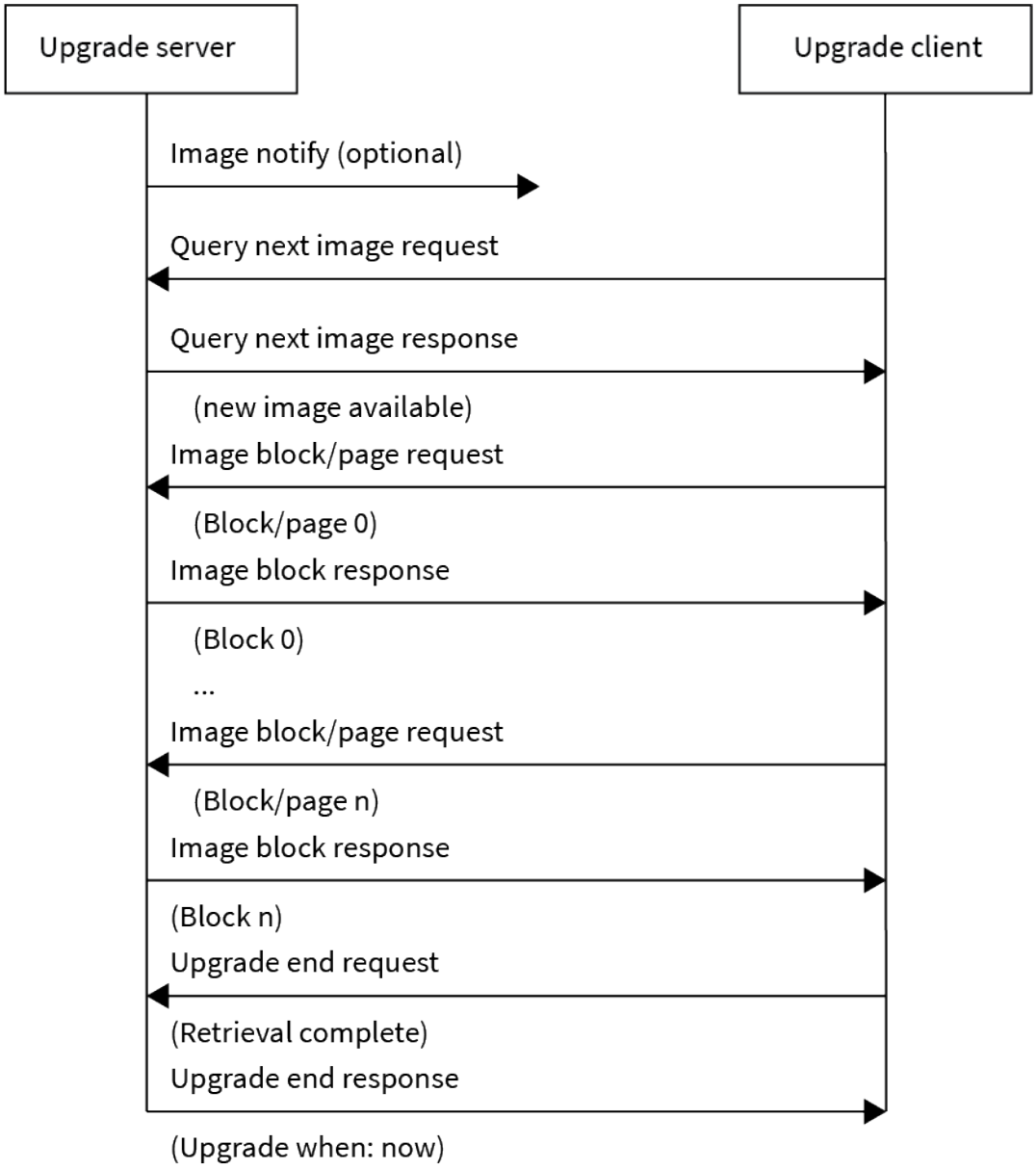
Offset	Length	Field name	Description
0	2	Sub-element tag	The tag for the sub-element, in little-endian format. This is usually 0x0000 for 'upgrade image'—this is the case for both firmware upgrades and file system upgrades.
2	4	Sub-element length	The length of the sub-element data (n) in little-endian format.
6	n	Sub-element data	The data to be transferred. This is either the contents of a .gbl firmware image or a signed file system image.

OTA upgrade process

The OTA upgrade process is performed by sending OTA commands between the client and server. OTA commands are sent as explicitly addressed packets, as described in OTA commands.

To initiate an OTA upgrade, the upgrade server sends an Image Notify Command, either to a single device or as a broadcast. After that initial transmission, the OTA process is driven by the client—or clients, if the Image Notify command is sent as a broadcast and accepted by multiple clients. The client sends requests to the server to request the image information, download it, and request when to upgrade. If the client does not receive a response from the server, it retries its request a few times before aborting the upgrade. The requests sent by the client are designed so that the server does not have to store any state related to a client's upgrade in progress—it only needs to send the image notify and respond to requests as they come in. The server can still observe these requests to track the state of an upgrade if desired, however—for example, to report download progress.

The following diagram shows the sequence of transmissions for an OTA upgrade:



OTA commands

All OTA commands are sent as explicitly addressed packets with the following address information:

- **Source/destination endpoint:** 0xE8
- **Cluster ID:** 0x0019
- **Profile ID:** 0xC105

The first three payload bytes of the command indicate what the command is and the structure of the remaining data in the command. All integer values in OTA commands are represented using little-endian byte order.

Image Notify command

(see ZCL Spec §11.13.3)

The Image Notify command is sent by the server to alert clients that an upgrade is available and prompt them to begin the upgrade. This command can be sent either as a broadcast or as a unicast:

- If sent as a unicast, the client will respond with a Query Next Image Request if the Image Notify contains valid information, and with a default response otherwise.
- If sent as a broadcast, all receiving clients will examine any optional fields included and respond only if the information indicates an image compatible with that device. On large networks, the query jitter parameter can be used to make only a percentage of those receiving the command respond at a time.

ZCL command format

Offset	Length	Field Name	Description
0	1	Frame control	When sending this command, value to set depends on whether the command will be sent as a broadcast or a unicast: ■ if sending a unicast: set this field to 0x09 (server-to-client command). ■ if sending a broadcast: set this field to 0x19 (server-to-client command, Default Response disabled).
1	1	Sequence number	Any sequence number can be used for the Image Notify
2	1	Command ID	0x00 for Image Notify
3	1	**Payload type**	Indicates which fields are present: 0: No optional fields (Query Jitter only) 1: Query Jitter, Manufacturer Code 2: Query Jitter, Manufacturer Code, Image Type 3: Query Jitter, Manufacturer Code, Image Type, File Version
4	1	**Query jitter**	A number, 0-100, must be set to 100 for a unicast. If less than 100 for a broadcast, then each receiving device will generate a random number and only respond to this command if that generated number is less than the query jitter.

Offset	Length	Field Name	Description
5	2	**Manufacturer code**	Optional. The Manufacturer code for the available image, parsed from the OTA file header.
7	2	**Image type**	Optional. The image type of the available image, parsed from the OTA file header.
9	4	**New file version**	Optional. The version parsed from the available image's OTA file header.

Example

To send this command from a server device, use the following Explicit Addressing Command Request - 0x11:

```
7E 00 21 11 01 00 13 A2 00 11 22 33 44 FF FE E8 E8 00 19 C1 05 00 00 09 01 00 03 64 1E 10 00 00 0A 20 00 01 18
```

The payload portion of the API frame (starting at offset 23) is shown below:

	Frame control	Sequence number	Command ID	Payload type	Query jitter	Manufacturer code	Image type	New file version
Data	09	01	00	03	64	1E 10	00 00	0A 20 00 01
Value	0x09	0x01	0x00	0x03	0x64 (100)	0x101E	0x0000	0x0100200A
Description			Image Notify	All fields present	Client will always respond	Digi's manufacturer code	Firmware upgrade	Must match value in the OTA file header. 0x01: Software compatibility number 0x00200A: Application version

Additional error cases

If a client receives a unicast Image Notify command that includes any optional fields—Manufacturer ID, Image Type, New File Version—and those fields do not match what the client is expecting, it will send a default response to the server. See Default Response command for more information on possible error cases.

Query Next Image Request command

(See ZCL Spec §11.13.4)

The Query Next Image Request command is sent by the client to ask for information on any available OTA Upgrade. It is sent in response to an Image Notify from the server.

ZCL command format

Offset	Length	Field Name	Description
0	1	Frame control	Will be set to 0x01, indicating a client to server command.
1	1	Sequence number	Sequence number chosen by the client.
2	1	Command ID	0x01 for Query Next Image Request.
3	1	**Field control**	Indicates which optional fields are present.
4	2	**Manufacturer code**	Manufacturer code of the client.
6	2	**Image type**	Image type that the client is requesting: ■ 0x0000 for a firmware upgrade ■ 0x0100 for a file system upgrade
8	4	**Current file version**	Firmware version that is currently running on the client. See File version definition for more information on how to interpret this field. **Note** The compatibility number reported in the current file version field refers to the installed firmware's compatibility number, which may be different from the **%C** value of the device.
12	2	**Hardware version**	Optional. Hardware version of the client.

Example

This is an example Explicit Rx Indicator (0x91) frame containing a Query Next Image Request that could be received by a server:

```
7E 00 1E 91 00 13 A2 00 55 66 77 88 FF FE E8 E8 00 19 C1 05 01 01 02 01 00 1E 10 00 00 06 20 00 01 F9
```

The payload portion of the API frame (starting at offset 21) is shown below:

	Frame control	Sequence number	Command ID	Field control	Manufacturer code	Image type	Current version
Data	01	02	01	00	1E 10	00 00	06 20 00 01
Value	0x01	0x02	0x01	0x00	0x101E	0x0000	0x01002006
Description			Query Next Image Request	HW version not present	Digi's manufacturer code	Firmware upgrade	0x01: Software compatibility number 0x002006: Application version

Query Next Image Response command

(See ZCL Spec §11.13.5)

The Query Next Image Response command should be sent by the server when it receives a Query Next Image request.

ZCL command format

Offset	Length	Field Name	Description
0	1	Frame control	Should be set to 0x19, indicating a server-to-client command.
1	1	Sequence number	Must match the sequence number of the request that prompted this response.
2	1	Command ID	0x02 for Query Next Image Response.
3	1	**Status**	One of three values: ■ 0x00 (SUCCESS): An image is available ■ 0x98 (NO_IMAGE_AVAILABLE): No upgrade image is available ■ 0x7E (NOT_AUTHORIZED): This server isn't authorized to perform an upgrade Remaining fields are only included if this field contains 0x00 (SUCCESS).
4	2	**Manufacturer code**	The Manufacturer code for the available image, parsed from the OTA file header. Must match the manufacturing code from the Query Next Image request that prompted this response.

Offset	Length	Field Name	Description
6	2	**Image type**	The Image for the available image, parsed from the OTA file header. Must match the manufacturing code from the Query Next Image request that prompted this response.
8	4	**File version**	The version parsed from the available image's OTA file header.
12	4	**Image size**	The size in bytes of the image that will be sent over the air. This should be the size of the OTA file. **Note** This field is handled differently if the client has a firmware version older than 200A. See Does the download include the OTA header?.

Example

An OTA server could respond to the Query Next Image Request example in the previous section using the following Explicit Addressing Command Request - 0x11:

```
7E 00 24 11 01 00 13 A2 00 11 22 33 44 FF FE E8 E8 00 19 C1 05 00 00 19 02 02 00 1E 10 00 00 0A 20 00 01 3A 90 05 00 9D
```

The payload portion of the API frame (starting at offset 23) is shown below:

	Frame Control	Sequence Number	Command ID	Status	Manufacturer Code	Image Type	File Version	Image Size
Data	19	02	02	00	1E 10	00 00	0A 20 00 01	3A 90 05 00
Value	0x19	0x02	0x02	0x00 (SUCCESS)	0x101E	0x0000	0x0100200A	0x0005903A
Description					Digi's manufacturer code	Firmware upgrade	Must match value in the OTA file header. 0x01: Software compatibility number 0x00200A: Application version	

This indicates that the server has version 0x0100200A available for the client to upgrade to, and that the file's size is 0x0005903A (364,6042) bytes.

Image Block Request command

(See ZCL Spec §11.13.6)

The client sends Image Block Request commands to the server to download the upgrade image data. The client will send requests until it has downloaded the entire image, as determined by the image size given in the Query Next Image Response from the server.

ZCL command format

Offset	Length	Field Name	Description
0	1	Frame control	Will be set to 0x01, indicating a client to server command.
1	1	Sequence number	Sequence number chosen by the client.
2	1	Command ID	0x03 for Image Block Request.
3	1	**Field control**	Indicates which optional fields are present. No optional fields are currently used by the XBee 3 802.15.4 RF Module.
4	2	**Manufacturer code**	The manufacturer code of the image being downloaded.
6	2	**Image type**	The image type of the image being downloaded.
8	4	**File version**	The version number of the file being downloaded.
12	4	**File offset**	The offset at which to begin the data, from the start of the OTA file. **Note** This field is handled differently if the client has a firmware version older than 200A. See Does the download include the OTA header?
13	1	**Maximum data size**	The maximum number of bytes of image data the server may include in its response.

Note Optional fields have been omitted here as they are not used by the XBee 3 802.15.4 RF Module.

Example

This is an example Explicit Receive Indicator - 0x91 containing an Image Block Request that could be received by a server:

```
7E 00 25 11 01 00 13 A2 00 11 22 33 44 FF FE E8 E8 00 19 C1 05 00 00 01 12 03 00 1E 10 00 00 0A 20 00 01 34 12 00 00 63 CA
```

The payload portion of the API frame (starting at offset 21) is shown below:

	Frame control	Sequence number	Command ID	Field control	Manufacturer code	Image type	Current version	File offset	Maximum data size
Data	01	12	03	00	1E 10	00 00	0A 20 00 01	34 12 00 00	63
Value	0x01	0x12	0x01	0x00	0x101E	0x0000	0x0100200A	0x00001234	0x63
Description			Image Block Request	No optional fields present	Digi's manufacturer code	Firmware upgrade	0x01: Software compatibility number 0x00200A: Application version		

The client is requesting up to 0x63 bytes of data, starting from offset 0x1234.

Image Block Response command

(See ZCL Spec §11.13.8)

The Image Block Response is generated by the OTA server to send the data asked for in an Image Block Request.

ZCL command format

Offset	Length	Field Name	Description
0	1	Frame control	Should be set to 0x19 indicating a server-to-client command.
1	1	Sequence number	Must match the sequence number of the request that prompted this response.
2	1	Command ID	0x05 for Image Block Response.

Offset	Length	Field Name	Description
3	1	**Status**	This field has one of two values, and determines the structure of the remaining fields: ■ 0x00 (SUCCESS): Image data is available. The remaining fields must be included. ■ 0x95 (ABORT): Instructs the client to abort the download. The remaining fields must not be included. **Note** The 0x97 (WAIT_FOR_DATA) status (see ZCL Spec §11.13.8.1) is not supported.
4	2	**Manufacturer code**	The Manufacturer code for the available image, parsed from the OTA file header. Must match the manufacturing code from the request that prompted this response.
6	2	**Image type**	The Image for the available image, parsed from the OTA file header. Must match the manufacturing code from the request that prompted this response.
8	4	**File version**	The version parsed from the available image's OTA file header. Must match the version number from the request that prompted this response.
12	4	**File offset**	The offset into the OTA file where the data begins. Must match the offset from the request that prompted this response. **Note** This field is handled differently if the client has a firmware version older than 200A. See Does the download include the OTA header?
16	1	**Data size**	The number of bytes of data included in this block. This can be any number less than or equal to the maximum data size value in the request that prompted this response.
17	n	**Image data**	Image data starting from the given offset. The length of this field is determined by the value in the preceding field (Data Size).

Example

An OTA server could respond to the Image Block Request example in the previous section using the following Explicit Addressing Command Request - 0x11:

```
7E 00 28 11 01 00 13 A2 00 11 22 33 44 FF FE E8 E8 00 19 C1 05 00 00 19 12 05 00 1E 10 00 00 0A 20 00 01 34 12 00 00 03 69
6D 67 D3
```

The payload portion of the API frame (starting at offset 23) is shown below:

	Frame control	Sequence number	Command ID	Status	Manufacturer code	Image type	File version	File offset	Data size	Image data
Data	19	12	05	00	1E 10	00 00	0A 20 00 01	34 12 00 00	03	69 6d 67
Value	0x19	0x12	0x05	0x00 (SUCCESS)	0x101E	0x0000	0x0100200A	0x00001234	0x03	69 6d 67
Description			Image Block Response		Digi's manufacturer code	Firmware upgrade	0x01: Software compatibility number 0x00200A: Application version			

This response contains three bytes of data starting at offset 0x1234. The data size value in this example is very small—three bytes—for simplicity; since any size less than or equal to the client's requested maximum is allowed this is a valid frame, but smaller image blocks will increase the time the OTA upgrade takes.

Upgrade End Request command

(See ZCL Spec §11.13.9)

The Upgrade End Request command is sent by the client when it finishes a download, whether successfully or not.

ZCL command format

Offset	Length	Field Name	Description
0	1	Frame control	Will be set to 0x01, indicating a client to server command.
1	1	Sequence number	Sequence number chosen by the client.
2	1	Command ID	0x06 for Upgrade End Request.

Offset	Length	Field Name	Description
3	1	**Status**	One of four values indicating the status of the download. ■ 0x00 (SUCCESS): The client successfully downloaded and verified the image. ■ 0x96 (INVALID_IMAGE): The client aborted the download because the downloaded image was invalid or corrupted. ■ 0x95 (ABORT): The client aborted the download for another reason. ■ 0x99 (REQUIRE_MORE_IMAGE): The download completed, but additional files are needed for the upgrade. This status is not used by the XBee 3 802.15.4 RF Module. The value of this field determines what response the server should send. If the status is 0x00 (SUCCESS), the server should respond with an Upgrade End Response command. Otherwise, the server should respond with a Default Response command with the SUCCESS status.
4	2	**Manufacturer code**	The manufacturer code of the image being downloaded.
6	2	**Image type**	The image type of the image being downloaded.
8	4	**File version**	The version of the image being downloaded

Exampe

This is an example Explicit Receive Indicator - 0x91 containing an Upgrade End Request that could be received by a server:

```
7E 00 1E 91 00 13 A2 00 55 66 77 88 FF FE E8 E8 00 19 C1 05 01 01 95 06 00 1E 10 00 00 0A 20 00 01 5D
```

The payload portion of the API frame (starting at offset 21) is shown below:

	Frame control	Sequence number	Command ID	Status	Manufacturer code	Image type	File version
Data	01	95	06	00	1E 10	00 00	0A 20 00 01
Value	0x01	0x95	0x06	0x00 (SUCCESS)	0x101E	0x0000	0x0100200A

	Frame control	Sequence number	Command ID	Status	Manufacturer code	Image type	File version
Description			Upgrade End Request		Digi's manufacturer code	Firmware upgrade	0x01: Software compatibility number 0x00200A: Application version

The client has completed the download of version 0x0100200A. The server should respond with an Upgrade End Response command.

Upgrade End Response command

(See ZCL Spec §11.13.9.6)

The Upgrade End Response command is sent by the server when it receives an Upgrade End Request with the SUCCESS status. This command instructs the device to perform the upgrade, and can be used to schedule an upgrade for a later time. An Upgrade End Response can also be sent without a request from a client if the client is waiting for an upgrade—scheduled by a previous Upgrade End Response—to change the time to wait for that upgrade.

ZCL command format

Offset	Length	Field Name	Description
0	1	Frame control	Should be set to 0x19 indicating a server-to-client command.
1	1	Sequence number	If this command is sent in response to an Upgrade End request, the sequence number should match the one from that request.
2	1	Command ID	0x07 for Upgrade End Response.
3	2	**Manufacturer code**	The Manufacturer code for the available image, parsed from the OTA file header. Must match the manufacturer code from the request that prompted this response.
5	2	**Image type**	The Image for the available image, parsed from the OTA file header. Must match the image type from the request that prompted this response.
7	4	**File version**	The version parsed from the available image's OTA file header. Must match the version number from the request that prompted this response.

Offset	Length	Field Name	Description
11	4	**Current time**	The current time, used for scheduled upgrades. See Schedule an upgrade for more information.
15	4	**Upgrade time**	The scheduled upgrade time, used for scheduled upgrades. See Schedule an upgrade for more information.

If the upgrade should be performed immediately and not scheduled for a later time, the Current Time and Upgrade Time fields should be set to the same value less than 0xFFFFFFFF.

Example

An OTA server could respond to the Image Block Request example in the previous section using the following Explicit Addressing Command Request - 0x11:

```
7E 00 27 11 01 00 13 A2 00 11 22 33 44 FF FE E8 E8 00 19 C1 05 00 00 19 95 07 1E 10 00 00 0A 20 00 01 00 00 00 00 00 00 00
00 D4
```

The payload portion of the API frame (starting at offset 23) is shown below:

	Frame control	Sequence number	Command ID	Manufacturer code	Image type	File version	Current time	Upgrade time
Data	19	95	07	1E 10	00 00	0A 20 00 01	00 00 00	00 00 00 00
Value	0x19	0x95	0x07	0x101E	0x0000	0x0100200A	0x00000000	0x00000000
Description			Upgrade End Response	Digi's manufacturer code	Firmware upgrade	0x01: Software compatibility number 0x00200A: Application version		

With the current time and upgrade time both set to 0, the device will reboot and install the upgrade immediately.

Default Response command

(See ZCL Spec §2.5.12)

A Default Response command is sent when a response is needed but there is no other command frame suited to the response.

During the OTA Upgrade process, the client will send a default response with an error status if it receives an invalid command from the server. The only time the server needs to send a default response is when it receives an Upgrade End Request with an error status; the server responds with a default response with status 0x00 (SUCCESS) status to indicate that the request was received.

ZCL command format

Offset	Length	Field Name	Description
0	1	Frame control	If command is sent by the client: 0x10 If command is sent by the server: 0x18
1	1	Sequence number	Must match the sequence number of the command that prompted this Default Response.
2	1	Command ID	0x0B for Default Response.
3	1	**(Source) command identifier**	The command ID of the command that prompted this Default Response.
4	1	**Status code**	A status code indicating success or an error. A full list of status codes, see ZCL Spec §2.6.3.

Error messages sent by the client

The client will send a default response to the server when an error occurs. The significance of the status code in this message depends on what server command prompted the default response. The **Handling Error Cases** section of each command's section in the ZCL specification contains detailed information on what errors a command can produce. Some errors that can be sent by the client are listed below:

Source Command Identifier	Status	Description
0x00 (Image Notify)	0x80 (MALFORMED_ COMMAND)	Either one of the errors form ZCL Spec §11.13.3.5.1, or manufacturer code or image type is not valid.
	0x70 (REQUEST_ DENIED)	OTA Upgrades have been disabled on this device.
	0x8A (DUPLICATE_ EXISTS)	The new version is not valid: ■ For firmware upgrades, the new firmware version must be different than what is installed on the device. Upgrades to the same version are not supported. ■ For file system upgrades, the version indicates what firmware version the image supports. It must match the currently installed firmware. Make sure the firmware version in the Image Notify is being parsed from the OTA header in the upgrade image.
	0x85 (INVALID_ FIELD)	Firmware is incompatible with the client's **%C (Hardware Compatibility)** value.
0x02 (Query Next Image Response)	0x80 (MALFORMED_ COMMAND)	The format of the command is invalid (see ZCL Spec §11.13.5.5).
	0x89 (INSUFFICIENT_ SPACE)	The image is too large for the client to store.
0x05 (Image Block Response)	0x80 (MALFORMED_ COMMAND)	The format of the command is invalid (See ZCL Spec §11.13.8.5).
0x07 (Upgrade End Response)	0x80 (MALFORMED_ COMMAND)	The format of the command is invalid (See ZCL Spec §11.13.9.9).

Example

After unicasting an Image Notify command to a client, the server may receive the following Explicit Receive Indicator - 0x91 frame containing a Default Response:

```
7E 00 17 91 00 13 A2 00 55 66 77 88 FF FE E8 E8 00 19 C1 05 01 10 0C 0B 00 8A A1
```

The payload portion of the API frame (starting at offset 21) is shown below:

	Frame control	Sequence number	Command ID	Source command identifier	Status
Data	10	0C	0B	00	8A
Value	0x10	0x0C	0x0C	0x00	0x8A (DUPLICATE_EXISTS)
Description			Default Response	Image Notify	

The source command identifier field indicates that the error is in response to an image notify, and the sequence number will match that of the Image Notify command sent by the server. According to the table above, a DUPLICATE_EXISTS status for an Image Notify means that the firmware version is invalid—the device is already running the firmware version that the server is trying to send.

When the server needs to send a default response, it can do so using an Explicit Addressing Command Request - 0x11. For example, to send a Default Response with a SUCCESS status in response to an Upgrade End Request:

```
7E 00 19 11 01 00 13 A2 00 11 22 33 44 FF FE E8 E8 00 19 C1 05 00 00 18 41 0B 06 00 78
```

The payload portion of the API frame (starting at offset 23) is shown below:

	Frame control	Sequence number	Command ID	Source command identifier	Status
Data	18	41	0B	06	00
Value	0x18	0x41	0x0B	0x06	0x00 (SUCCESS)
Description			Default Response	Upgrade End Response	

Handling unrecognized commands

If the server receives a command with an unrecognized command ID, it should respond with a default response with status 0x81 (UNSUP_CLUSTER_COMMAND).

Schedule an upgrade

The current time and upgrade time fields of the Upgrade End Response command can be used to schedule an upgrade for some time in the future. The time can for the upgrade can be scheduled in several ways:

Current time value	Upgrade time value	Effect
0x00000000-0xFFFFFFFE	Equal to current time	The device will upgrade immediately.
0x00000000	0x00000001-0xFFFFFFFE	Delayed upgrade: the device will upgrade after the number of seconds indicated by the upgrade time value.
0x00000001-0xFFFFFFFE (Current time in seconds since midnight Jan 1, 2000)	Any value greater than current time and less than 0xFFFFFFFF (Intended upgrade time in seconds since midnight Jan 1, 2000)	Scheduled upgrade: the device will determine how long to wait by subtracting current time from upgrade time, and wait that long before upgrading.
Any	0xFFFFFFFF	Prompted upgrade: The device will not upgrade, and will wait indefinitely to receive another Upgrade End Response with the server. The second upgrade end response can schedule an upgrade with any of the above methods.

Note When performing a scheduled upgrade, we recommend that the OTA upgrade server continue to monitor for and respond to OTA commands until after the time the upgrade is meant to be applied. If the client loses power while waiting to apply a scheduled upgrade, it will send another Upgrade End Request to the server when it regains power in an attempt to resume the schedule. If the client does not receive a response from the server after a few tries, it applies the upgrade without confirmation from the server.

Scheduled upgrades on sleeping devices

To schedule an upgrade, an XBee 3 802.15.4 RF Module makes use of internal software timers, which only count time while the device is awake. So a sleeping device takes significantly longer to apply the scheduled upgrade than a non-sleeping device. Consider this limitation when scheduling an upgrade on a sleeping device.

Formula for estimating when a sleeping device will apply an upgrade

upgrade_delay = number of seconds the upgrade was scheduled for (**upgradeTime**- **currentTime** fields in the Upgrade End Response frame)

sleep_time = amount of time the device is estimated to be asleep (**SP** for an asynchronous sleeping device)

wake_time = amount of time the device is estimated to be awake (**ST** for an asynchronous sleeping device)

total_time = sleep_time + wake_time

expected_upgrade_delay = upgrade_delay * (total_time / wake_time)

Asynchronous cyclic sleep scheduled upgrades

A device that is configured for asynchronous cyclic sleep will only be awake for a few milliseconds at a time, therefore we do not recommend that you schedule an upgrade for a sleeping node with this configuration. However, if the device is configured to always stay awake for **ST** time then the scheduled upgrade can be estimated by using the above formula—where wake_time = **ST** and sleep_time = **SP**. You can configure a device to always stay awake for **ST** by setting **SO** bit 8 to one—for example, **SO** = 0x80).

Pin sleep scheduled upgrades

Since the device only counts time while it is awake, scheduling an upgrade on a pin sleeping device may be unpredictable. However, if a pin sleeping device has predictable sleep patterns it is possible to estimate when a scheduled upgrade will be applied. The sleep estimate formula can be applied to a pin sleeping device to estimate when it will apply the upgrade.

Aggressively sleeping devices

If a device is asynchronously sleeping, and keeping it awake for all of **ST** time is undesired, then we recommend performing a scheduled upgrade in the following manner:

1. Configure the sleeping node for indirect messaging:
   a. Configure the sleeping device with the following parameters:
      - **CE** = **0** (join network)
      - **DH, DL** should be set to match **SH**, **SL** of the OTA server device
   b. Make sure that **ST** and **SP** of the sleeping device and OTA server radio match.
   c. Set all of the transmit option fields of the API frames sent to the OTA server device to **0x40**.
2. Download the firmware/file system image to the sleeping device as described in this section.
   a. When sending the Upgrade End Response frame set the **upgradeTime** to **0xFFFFFFFF**—instructing the sleeping device to wait for another upgrade end request before applying the upgrade.
3. Wait for the desired amount of time to pass.
4. When the time to have the sleeping device apply its upgrade has arrived, send a second Upgrade End Response to the sleeping device with the **currentTime** and **upgradeTime** fields both set to **0x0000**. This causes the sleeping device to apply the upgrade immediately.

Considerations for older firmware versions

Some changes need to be made to this OTA upgrade process for some previous versions of the software.

All versions older than 200A

- When the firmware is sent over the air it must be sent without including the OTA header and sub-element tags. See Does the download include the OTA header?
- These older versions will not retry requests; if a packet from the server is dropped, you may need to restart the upgrade.

Does the download include the OTA header?

Most OTA files consist of an OTA header, a sub-element tag, and a single sub-element: The upgrade image. For firmware versions 200A and newer, the entire OTA file is sent to the client during an OTA Upgrade. However, for versions older than 200A, only the contents of the file's single sub-element should be sent—not the OTA header or the sub-element tag. This affects several fields in the upgrade process.

When dealing with these two methods it is useful to know the **image offset** of the OTA file—that is, the offset at which the upgrade image data actually begins. This can be calculated by taking the size of the OTA header—which can be parsed from near the beginning of the OTA file—and adding six bytes for the sub-element header: two bytes for the tag, four bytes for the length.

Command	Field	Value when sending without header (pre-200A)	Value when sending with header (200A and later)	Notes
Query Next Image Response	Image size	The size of the upgrade image parsed from the first sub-element tag's length value, or the total size of the OTA file minus the image offset.	The total size of the OTA file.	In either case, this is the total number of bytes that the client needs to download. This value should never be determined by reading the **Total Image Size** field from the OTA header, as that field contains incorrect information on most older firmware files.
Image Block	File offset	This refers to the offset from the start of the upgrade image data—add the image offset to this value to get the offset into the OTA file.	This refers to the offset into the OTA file.	

Note For compatibility with older OTA upgrade servers, newer firmware versions support both methods for a firmware upgrade. File system upgrades only support the method corresponding to the installed firmware version, as described above. We recommend using the newer method where possible to ensure compatibility with future releases.

OTA file system upgrades

After a FOTA update, all file system data and bundled MicroPython code is erased. To continue running code, a new file system needs to be sent to the device after the firmware update is complete. This section contains information on how to update the file system of remote devices over the air.

OTA file system update process

Since OTA file system updates are signed, remote devices must be configured so that they can validate incoming updates. To set up a network for OTA file system updates:

1. Generate a public/private Elliptic Curve Digital Signature Algorithm (ECDSA) signing key pair.
2. Using the generated public key, set FK (File System Public Key) on all devices that will receive OTA file system updates.

Note You cannot set **FK** remotely. You must either set **FK** before the XBee 3 802.15.4 RF Module is deployed, or else serial access to the device is needed to set it.

To perform an OTA file system update:

1. On a local device, create a copy of the file system that you want to send over the air.
2. Create an OTA file system image, signed using the private key generated previously.
3. Perform an OTA update using the created OTA file.

Note The local device used to create the file system image must have the same firmware version installed as the target device or the file system will be rejected. Use VR (Firmware Version) to check the version number on both the staging and target devices.

You can perform all of these steps automatically through XCTU or manually using other tools.

OTA file system updates using XCTU

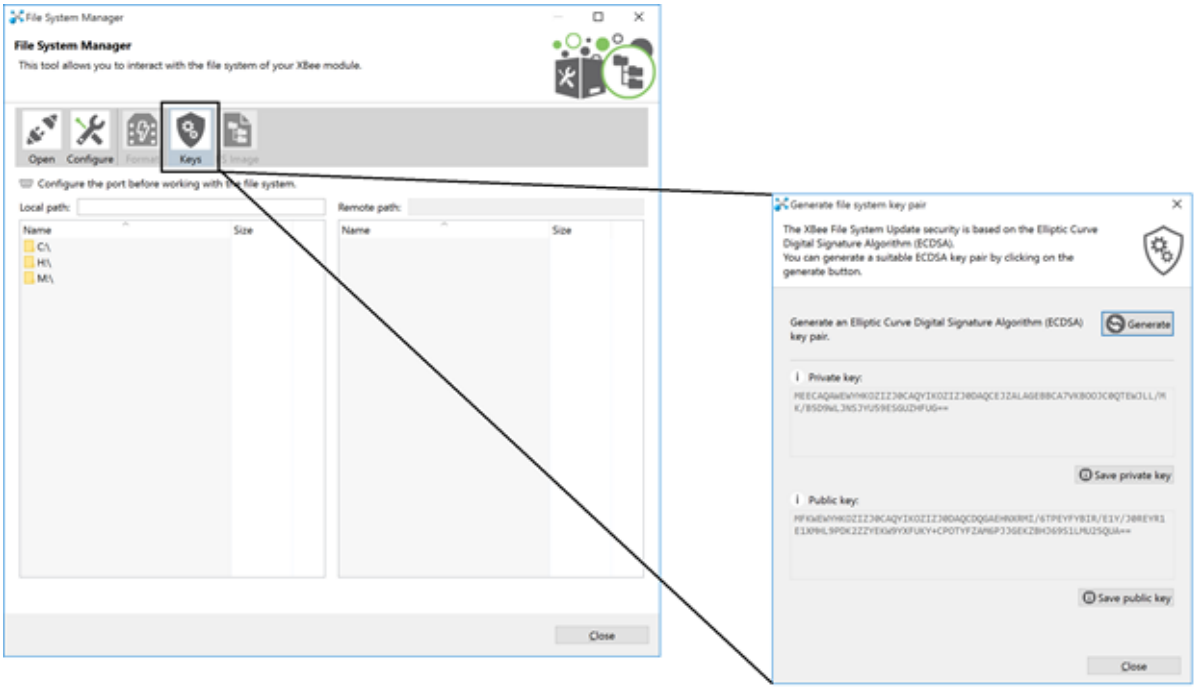
Use the following steps to perform a file system update OTA using XCTU:

1. Generate a public/private key pair
2. Set the public key on the XBee 3 device
3. Create the OTA file system image
4. Perform the OTA file system update

Generate a public/private key pair

XCTU provides an ECDSA key pair generator that you can use to store a public/private key pair in .pem files. To access the **Generate file system key pair** dialog:

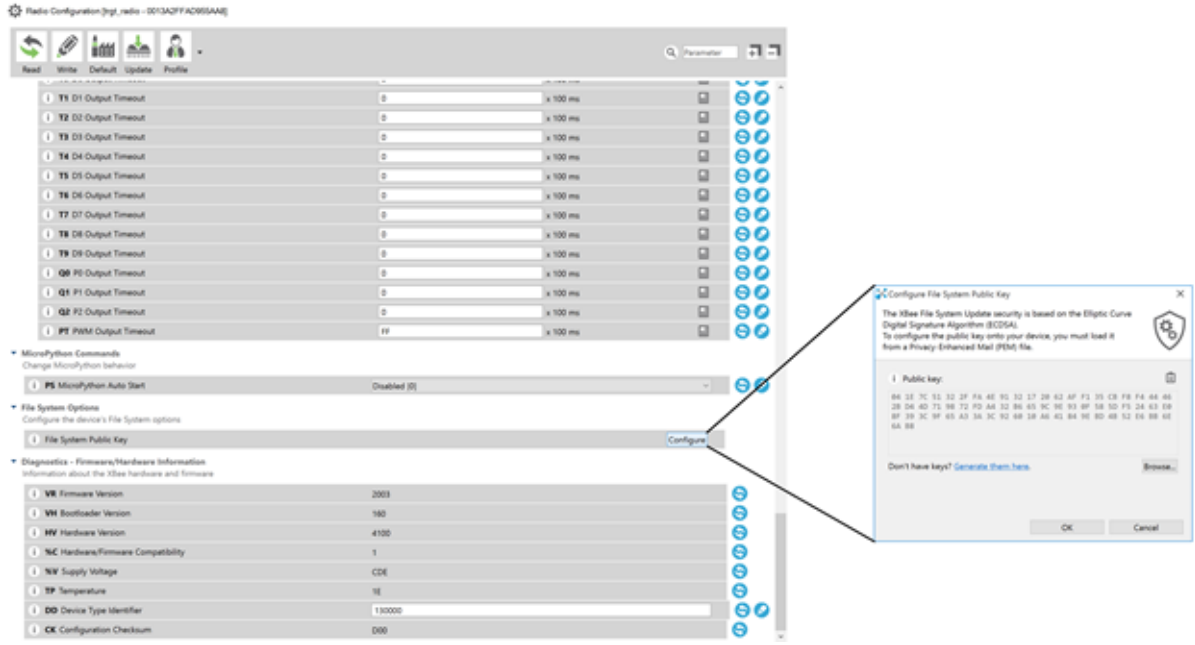
1. Open the **File System Manager** dialog box.
2. Click **Keys** as shown below.



3. Click **Generate** in the **Generate file system key pair** dialog.
4. Save both the keys in a safe location and close the dialog box.

Set the public key on the XBee 3 device

1. Open the configuration view of the target device in XCTU and go to the **File System** category.
2. In the **File System Public Key** row, click **Configure**.



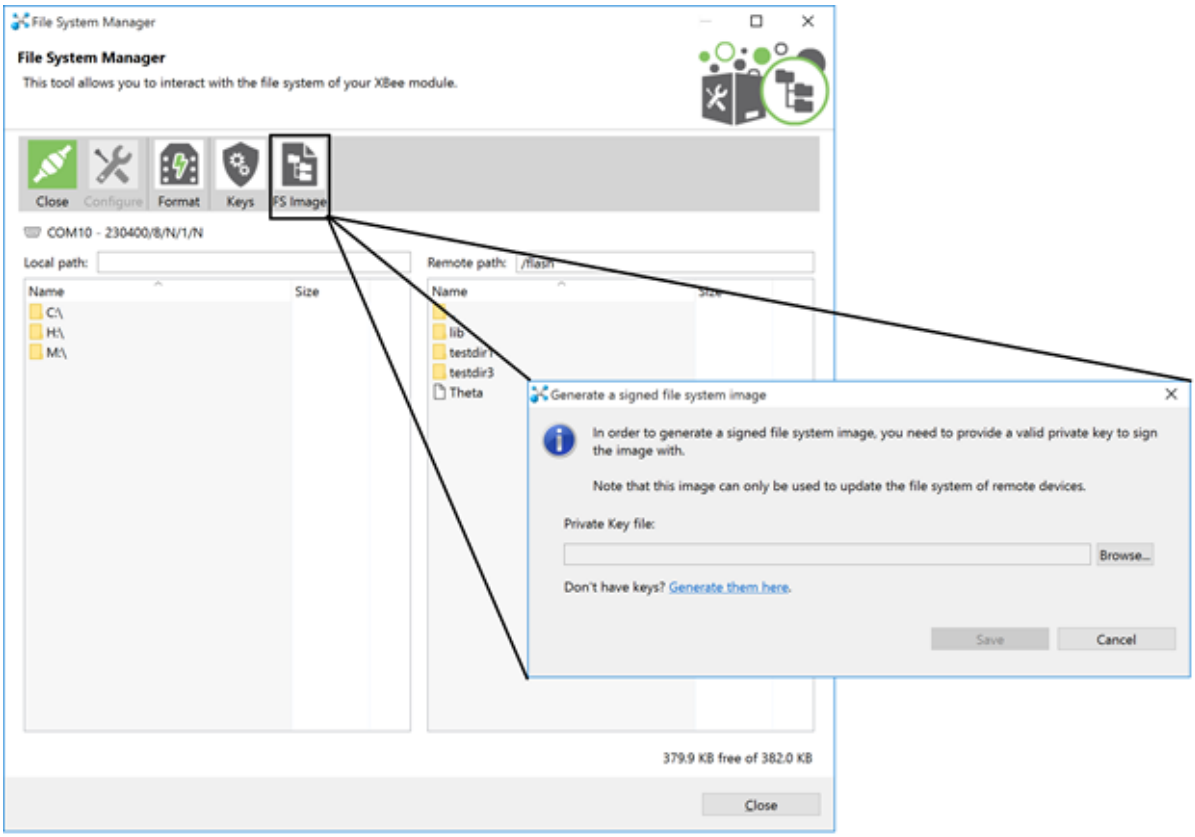
3. In the **Configure File System Public Key** dialog box, click **Browse** and choose the .pem file that you saved the public key into. Once this is done, the HEX value of the public key is visible under the **Public key** section on the dialog box as shown.
4. Click **OK** to ensure that the key gets written into the device.

Note This can be only be done locally. XBee 3 firmware **DOES NOT** support remotely setting the file system public key at this time.

Create the OTA file system image

To create the OTA file system image:

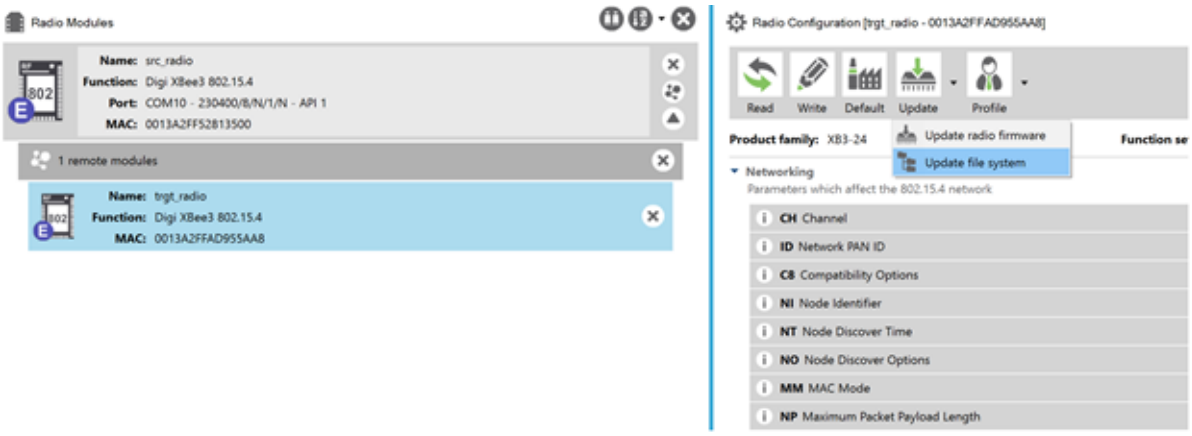
1. Open the **File System Manager** dialog box.
2. Open a connection on the device that you want to generate the OTA file system image from.
3. Click **FS Image**.
4. In the **Generate a signed file system image** window that displays, click **Browse** and choose the .pem file that the private key was stored in.
5. Once the path shows up on the **Private Key file** field, click **Save** to assign the .fs.ota an appropriate file name and location.
6. Save the file.



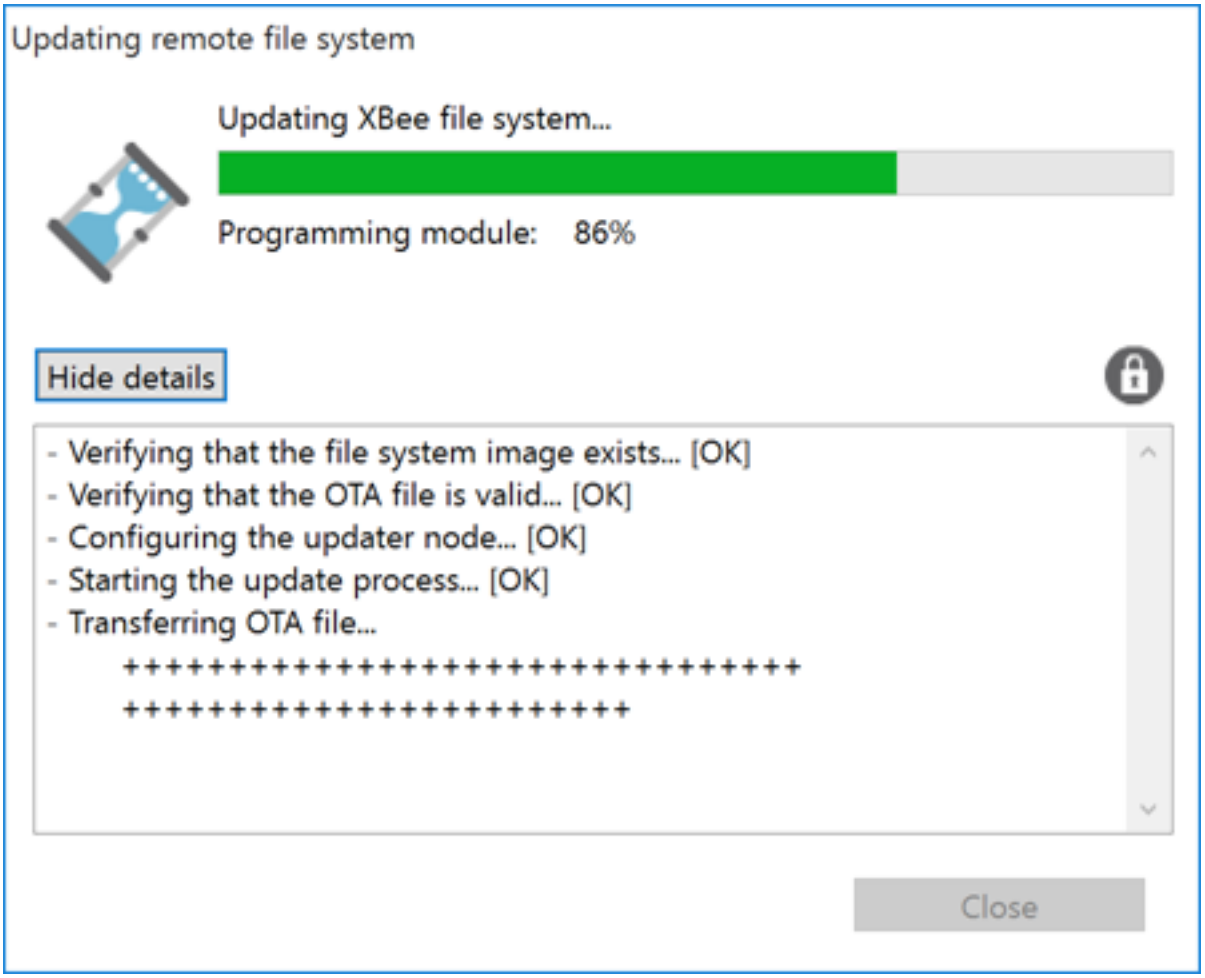
You will be prompted with a **File system image successfully saved** dialog box if the file was successfully generated.

Perform the OTA file system update

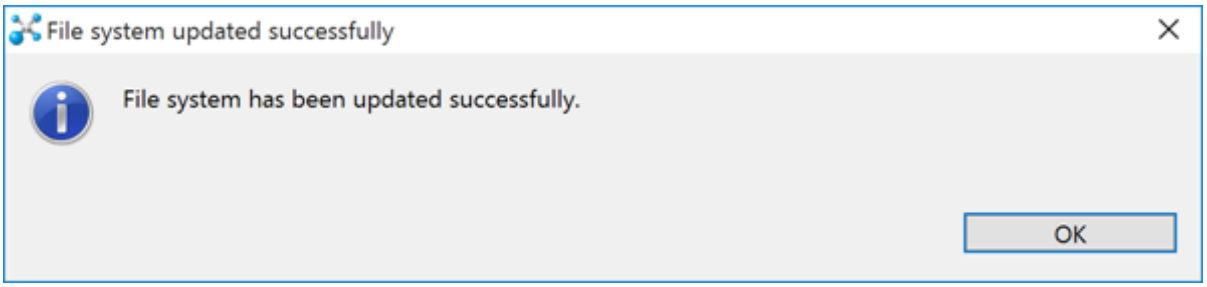
1. To add the target device, click **Discover radios in the same network** from the source device.
2. Enter Configuration mode on the remote device.
3. Click the down arrow next to the **Update** button and choose **Update File System**.



4. Choose the OTA file system image (.fs.ota) that the target node needs to be updated to.
5. Click **Open**.



Once the file system image is completely transferred and mounted on the remote device, XCTU informs you that the file system has been updated successfully.



OTA file system updates: OEM

Use the following steps to perform a file system update OTA using OEM tools:

1. Generate a public/private key pair
2. Set the public key on the XBee 3 device

3. Create the OTA file system image
4. Perform the OTA file system update

Generate a public/private key pair

Generate ECDSA signing keys using secp256r1 curve parameters (also known as prime256v1 or NIST P-256).

To generate a public/private key pair using OpenSSL, run the following command:

```
openssl ecparam -name prime256v1 -genkey -outform pem -out keypair.pem
```

To extract the private key from the key pair generated above:

```
openssl pkcs8 -topk8 -inform pem -in pair.pem -outform pem -nocrypt -out
private.pem
```

To extract the public key from the key pair generated above:

```
openssl ec -in keypair.pem -pubout -out public.pem
```

Set the public key on the XBee 3 device

The public keys generated by XCTU and OpenSSL are stored in *.pem files. These files need to be parsed to get the value to use when setting **FK**. To parse a public key file, run:

```
openssl asn1parse -in public.pem -dump
```

The command will produce something like the following output:

```
0:d=0  hl=2 l=  89 cons: SEQUENCE
  2:d=1  hl=2 l=  19 cons: SEQUENCE
  4:d=2  hl=2 l=   7 prim: OBJECT            :id-ecPublicKey
 13:d=2  hl=2 l=   8 prim: OBJECT            :prime256v1
 23:d=1  hl=2 l=  66 prim: BIT STRING
    0000 - 00 04 95 50 aa 55 b6 f5-5d 99 4d d8 15 d1 71 57   ...P.U..].M...qW
    0010 - 51 80 d5 14 ec 1f 6a 15-51 a2 c4 b8 0f 77 10 8a   Q.....j.Q....w..
    0020 - 33 a3 80 07 47 40 14 8b-5c a7 4c 78 02 fc 4d 82   3...G@..\.Lx..M.
    0030 - 90 4b 39 98 62 a1 1d 97-6e 78 fb 54 62 06 d2 41   .K9.b...nx.Tb..A
    0040 - c7 3b
```

The public key should be 65 bytes long - it is the BIT STRING value at the end, with the leading 00 omitted; in this case:

```
049550aa55b6f55d994dd815d171575180d514ec1f6a1551a2c4b80f77108a33a380074740148b5ca
74c7802fc4d82904b399862a11d976e78fb546206d241c73b
```

Create the OTA file system image

You can create a file system image outside of XCTU using any utility that can perform ECDSA signing. These instructions show how to do so using OpenSSL. To create an OTA file system image, use the following steps.

Create a staged file system

In order to create a usable file system image, first create a 'staged' copy of the file system you want to send on a local device.

Use the **FS** command or MicroPython to load all of the files that you want to send onto the local staging device.

Note The staging device must have the same firmware version installed as the target device or the file system will be rejected. Use the **VR** command to check the version number on both the staging and target devices.

Download the file system image

Run the command **ATFS GET /sys/xbfs.bin** to download an image of the file system from the staging device. The file is transferred using the YMODEM protocol. See File system for more information on downloading files using **FS GET**.

Pad the file system image

The file system image must be a multiple of 2048 bytes long before it is signed. Using hex editing software, add 0xFF bytes to the end of the downloaded image until size of the file is a multiple of 2048 (0x800 in hex).

Calculate the image signature

Once the image has been padded to a multiple of 2048 bytes, it is ready to be signed. The ECDSA signature should be calculated using SHA256 as the hash algorithm.

Assuming a public/private key pair has been generated as described in Generate a public/private key pair, that the private key is named private.pem, and that the padded image is named **xbfs.bin**; this can be done using OpenSSL with the following command:

```
openssl dgst -sha256 -sign private.pem -binary -out sig.bin xbfs.bin
```

sig.bin will contain the signature for the image.

Append the calculated signature to the image

The signature should be between 70 and 72 bytes, and it should be appended to the padded image.

Create the OTA file

Put the image into an OTA file that follows the format specified in ZigBee Document 095264r23. The file should consist of:

- An OTA header
- An upgrade image sub-element tag
- The padded, signed image data

The OTA file must begin with an OTA header. See The OTA header for information on the format of the header. The image type should be **0x0100** for a file system image upgrade.

The sub-element tag should come before the image data. The sub-element tag follows the format described in section **6.3.3** of ZigBee Document 095264r23. It consists of 6 bytes: the first 2 bytes are the tag id and should be set to **0x0000**. The next 4 bytes contain the length of the file system image in little-endian format.

Perform the OTA file system update

The process for performing an OTA file system update is the same as the process for performing a FOTA upgrade, as described in Over-the-air firmware/filesystem upgrade process for 802.15.4. Note

that the data that goes in the image blocks starts at the beginning of the image data, after the OTA header and sub-element tag.