Digi XBee® 3 ZigBee®

RF Module

User Guide

Revision history—90001539

Revision	Date	Description
G	April 2020	Added centralized trust center backups. Added the **BK**, **BP**, **CX**, **R?**, **US**, **RK**, and **KB** commands. Added OTA upgrades. **Added Open Join Window indication**.
H	May 2020	Revised all API frame descriptions. Added ZDO cluster commands and information.
J	September 2020	Added a note to **D8**. Updated **OTA firmware/file system upgrades**. Updated **Considerations for older firmware versions**.
K	April 2021	Updated **ZDO clusters** to indicate XBee support.
L	July 2021	Added safety instructions.

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 Zigbee® RF Module, 90001539 L) in the subject line of your email.

Contents

Secure access

File system

Get started with BLE

BLE reference

Serial communication

SPI operation

Modes

Zigbee networks

Transmission, addressing, and routing

Zigbee security

Centralized trust center backup

Manage End Devices

I/O support

AT commands

API Operation

Frame descriptions

OTA firmware/file system upgrades

OTA file system upgrades

Digi XBee® 3 Zigbee® RF Module

This manual describes the operation of the XBee 3 Zigbee RF Module, which consists of Zigbee firmware loaded onto XBee 3 hardware.

The XBee 3 Zigbee RF Modules provide wireless connectivity to end-point devices in Zigbee mesh networks. Using the Zigbee 3.0 feature set, these devices are inter-operable with other Zigbee devices, including devices from other vendors. With theXBee 3 Zigbee RF Module, users can have their Zigbee network up-and-running in a matter of minutes without configuration or additional development.

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

Applicable firmware and hardware

This user guide supports the following firmware:

- v.10xx Zigbee

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.

Safety instructions

XBee modules

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

Get started

Refer to the *XBee Zigbee Mesh Kit User Guide* for comprehensive instructions and examples on how to get started with the XBee 3 Zigbee RF Module.

Configure the XBee 3 Zigbee 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*.

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 Zigbee 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 Zigbee 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 Zigbee 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.

Software libraries

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

- XBee Java library
- XBee Python library

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

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

XBee Network Assistant

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

- Join and inspect any nearby XBee network to get detailed information about all the nodes it contains.
- Update the configuration of all the nodes of the network, specific groups, or single devices based on configuration profiles.
- Geo-locate your network devices or place them in custom maps and get information about the connections between them.

- Export the network you are inspecting and import it later to continue working or work offline.
- Use automatic application updates to keep you up to date with the latest version of the tool.

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

To install the XBee Network Assistant:

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

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.

Update the firmware over-the-air

The XBee 3 Zigbee RF Module supports firmware over-the-air (FOTA) updates. To perform an FOTA update, the device to be updated must be associated and communicable with a Zigbee network. In this section, the node performing the update is considered the server and the node being updated is the client.

Use XCTU to perform the FOTA update using the following process:

Add the device to XCTU

You must have a local device connected to your computer in order to perform firmware updates, either to update local firmware through the serial connection or to use the local device to remotely upgrade another device in the same network. With a local device properly attached to your computer, follow these steps:

1. Add the local device attached to your computer to XCTU so it displays in the radio modules list.
2. Add the remote module in the network to XCTU:
   a. Configure the local module you added to work in API mode.
   b. Click **Discover radio nodes in the same network** to start a search for the remote device.
   c. When a remote device is found, it is listed in the **Discovering remote devices** dialog.
   d. Select the device and click **Add selected devices**. The remote device is added to the radio modules list as a subordinate to the local device.

Update to the latest firmware

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

1. Click the **Configuration working modes** button .
2. Select a local or remote XBee module from the **Radio Modules** list.
3. Click the **Update firmware** button .

   The **Update firmware** dialog displays the available and compatible firmware for the selected XBee module.
4. Select the product family of the XBee module, the function set, and the latest firmware version.

Note XBee 3 Zigbee 3.0 does not support forced upgrades to the same version of the firmware.

5. Click **Update**. A dialog displays update progress. Click **Show details** for details of the firmware update process.

Note Once you add your device to the radio modules list in XCTU, the update process is exactly the same whether it is a local or remote device.

Note If there are instances where the upgrade fails with a transmission/waiting for image block request error, retry the update process.

See How to update the firmware of your modules in the XCTU User Guide for more information.

For information about performing a firmware over-the-air (FOTA) update outside of XCTU, see In-depth OTA firmware upgrade process for Zigbee 3.0.

Get started with MicroPython

This user guide provides an overview of how to use MicroPython with the XBee 3 Zigbee 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 Zigbee 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 Zigbee 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 Zigbee RF Module

The XBee 3 Zigbee RF Module has MicroPython running on the device itself. You can access a MicroPython prompt from the XBee 3 Zigbee 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 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 Zigbee RF Module installed on an appropriate development board such as an XBIB-U-DEV or an XBDB-U-ZB.
- The XBee 3 Zigbee 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 Zigbee 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 Zigbee 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 Zigbee 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 Zigbee 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 Zigbee 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 Zigbee 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 Zigbee 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 Zigbee 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 Zigbee 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 Zigbee RF Module.

Zigbee 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.
- MicroPython does not have support for all of the XBee API frame types, particularly for source routing. If you are planning to operate with a network of more than 40 nodes, Digi highly recommends that you operate with the aggregator in API mode and implement source routing.

For the examples in this section, we use MicroPython to manage a Zigbee network and send and receive data between modules. To follow the upcoming examples, we need to configure a second XBee 3 Zigbee 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.

WARNING! The upcoming examples form and join an unencrypted Zigbee network. If the modules were previously associated with a network, they will be disassociated.

Example: forming and joining a Zigbee network using MicroPython

This example forms a two-node Zigbee network using MicroPython. This is a prerequisite for subsequent networking examples.

This example assumes that you have two XBee 3 Zigbee RF Modules configured for MicroPython and two terminals open, one for each radio.

Execute the following code on the first radio; it will be our network coordinator:

```
import xbee, time
# Set the identifying string of the radio
xbee.atcmd("NI", "Coordinator")

# Configure some basic network settings
network_settings = {"CE": 1, "ID": 0xABCD, "EE": 0, "NJ": 0xFF}

for command, value in network_settings.items():
       xbee.atcmd(command, value)
xbee.atcmd("AC")  # Apply changes
time.sleep(1)

while xbee.atcmd("AI") != 0:
       time.sleep(0.1)
print("Network Established")

operating_network = ["OI", "OP", "CH"]
print("Operating network parameters:")
for cmd in operating_network:
       print("{}: {}".format(cmd, xbee.atcmd(cmd)))
```

Run the following code on the second radio, it will be a router that will join the established network:

```
import xbee, time

# Set the identifying string of the radio
xbee.atcmd("NI", "Router")

# Configure some basic network settings
network_settings = {"CE": 0, "ID": 0xABCD, "EE": 0}

for command, value in network_settings.items():
       xbee.atcmd(command, value)
xbee.atcmd("AC")  # Apply changes
time.sleep(1)
```

```
# Query AI until it reports success

print("Connecting to network, please wait...")
while xbee.atcmd("AI") != 0:
        time.sleep(0.1)
print("Connected to Network")

operating_network = ["OI", "OP", "CH"]
print("Operating network parameters:")
for cmd in operating_network:
        print("{}: {}".format(cmd, xbee.atcmd(cmd)))
```

After the code has been executed on both radios, the radio reports the operating network parameters. Make sure both radios report the same values to ensure they are on the same network.

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 Discovery).

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 Zigbee 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 Zigbee 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 Zigbee 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 a Zigbee 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 Zigbee network as a coordinator...")
xbee.atcmd("NI", "Coordinator")
network_settings = {"CE": 1, "ID": 0x3332, "EE": 0, "NJ": 0xFF}
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 router joins
```

```
        node_list = list(xbee.discover())
print("Remote node found, transmitting data")

for node in node_list:
        dest_addr = node['sender_nwk'] # using 16 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 router module

The following example code joins the Zigbee 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 a router...")
xbee.atcmd("NI", "Router")
network_settings = {"CE": 0, "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

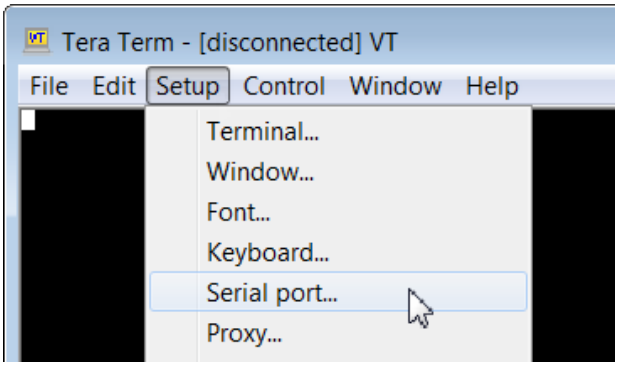
If you do not use the MicroPython terminal in XCTU, you can use other terminal programs to communicate with the XBee 3 Zigbee RF Module. If you use Microsoft Windows, follow the instructions for Tera Term; if you use Linux, follow the instructions for picocom. To download these programs:

- 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 Zigbee 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 Zigbee 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 Zigbee 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 Zigbee with EFR32MG
Type "help()" for more information.
>>>
```

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

Use picocom in Linux

With the XBee 3 Zigbee 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 Zigbee RF Module is connected to in order to communicate with the device.

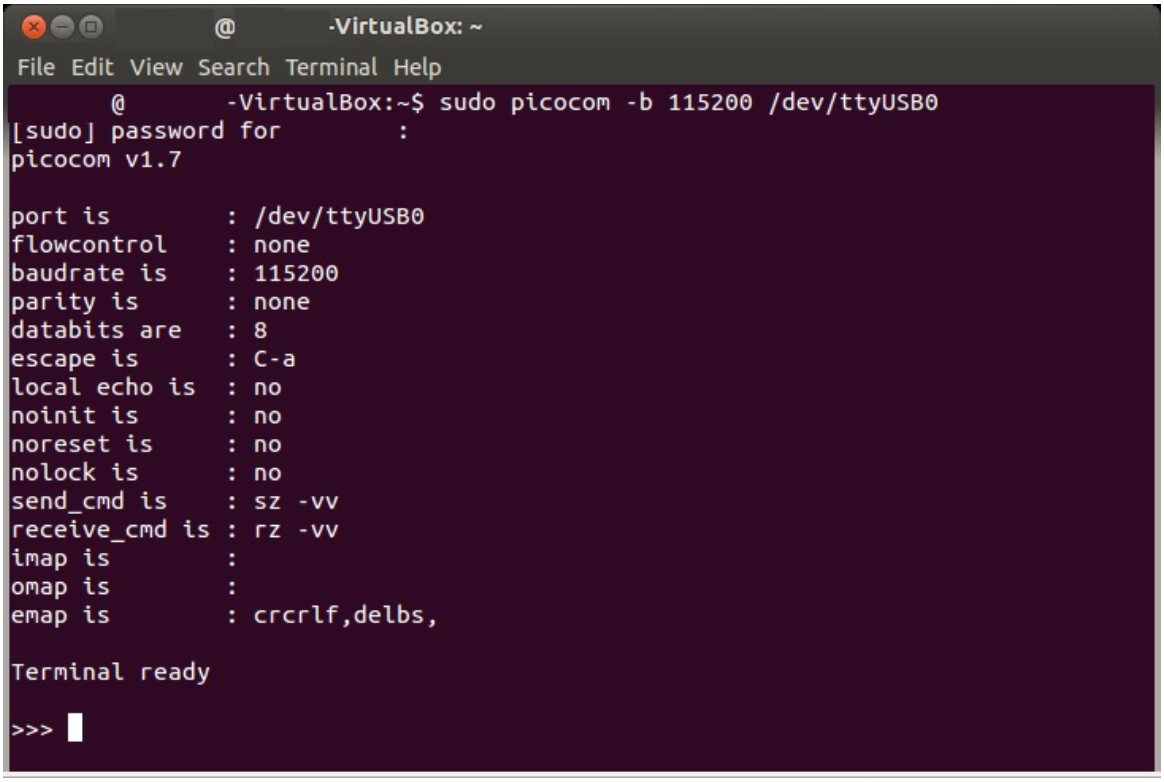
1. Open a terminal in Linux and type **picocom -b 115200 /dev/ttyUSB0**. This assumes you have no other USB-to-serial devices attached to the system.

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 Zigbee 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 Zigbee RF Module.
3. Check the directory again and you should see one additional device, which is the XBee 3 Zigbee 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-Zigbee
  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 Zigbee 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.

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.

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.

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.

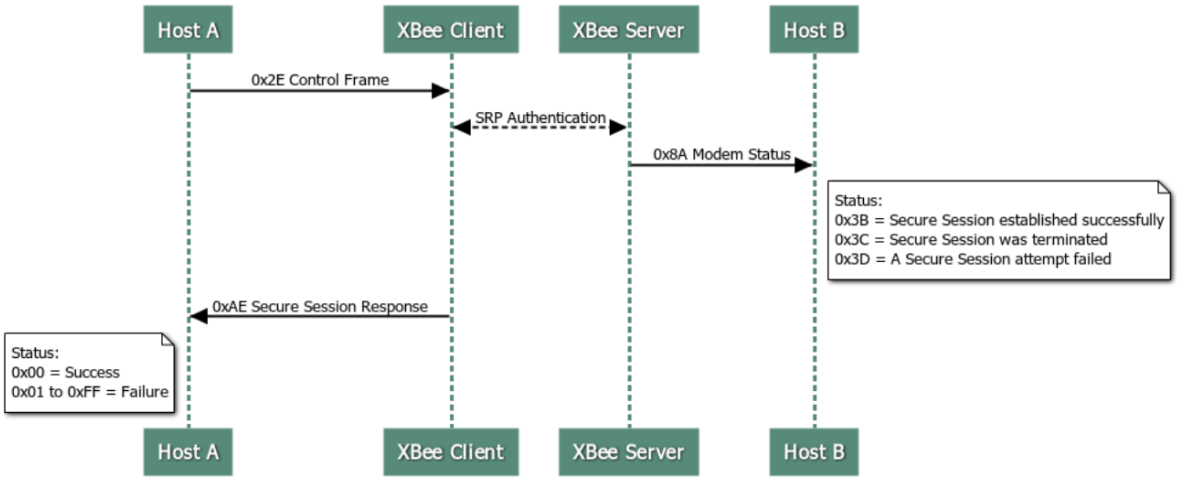
Secure Session API frames

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 Zigbee RF Module refer to the *Digi MicroPython Programming Guide*.

Overview of the file system

XBee 3 Zigbee RF Module firmware versions 1006 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 Zigbee 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 Zigbee 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 Zigbee 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 Zigbee RF Module

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

1. Set up the XBee 3 Zigbee 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 Zigbee 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 Zigbee 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 Zigbee 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.

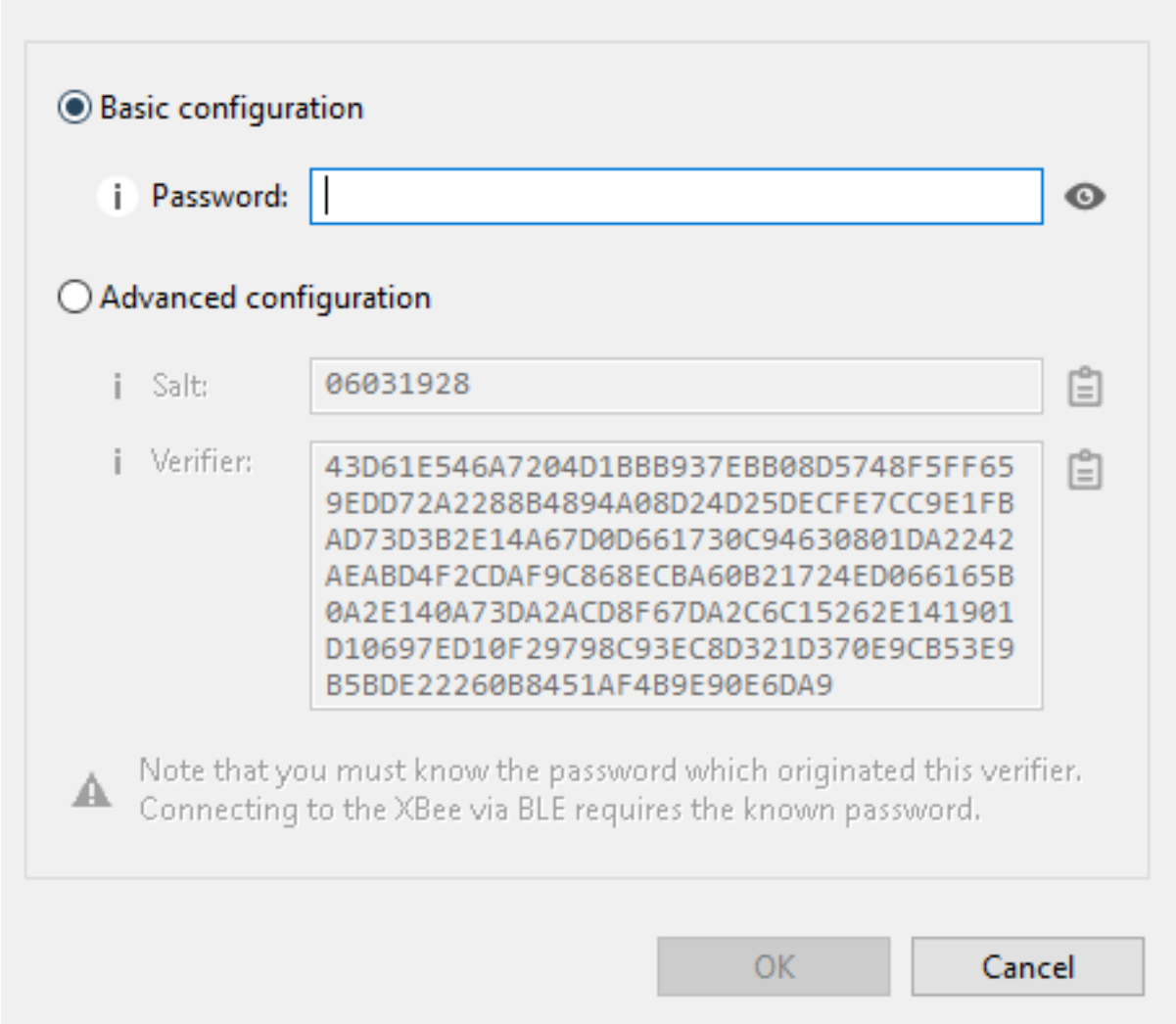
Note When you enter the BLE password in XCTU, the salt and verifier values are calculated as you set your password. For more information on how these values are used in the authentication process, see BLE Unlock Request - 0x2C.

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 Zigbee 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.

Serial communication

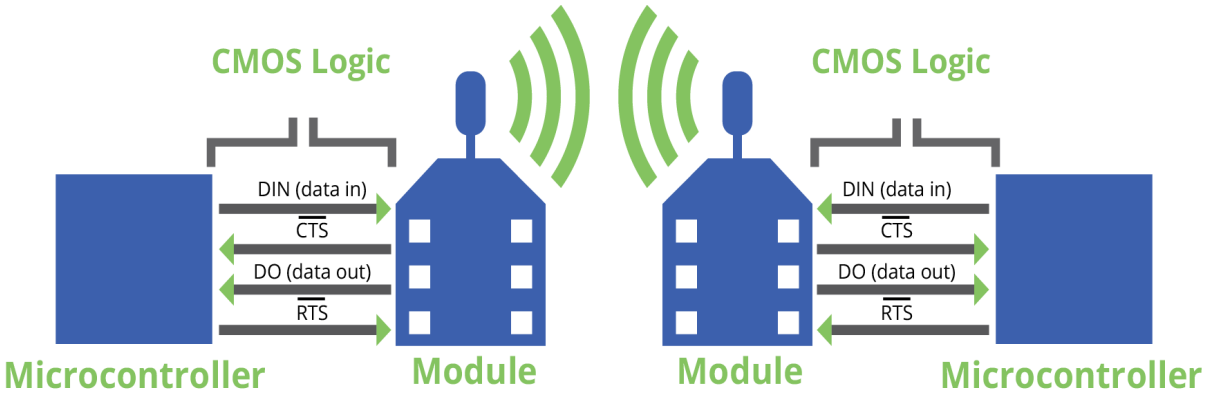
Serial interface

The XBee 3 Zigbee 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.

UART data flow

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



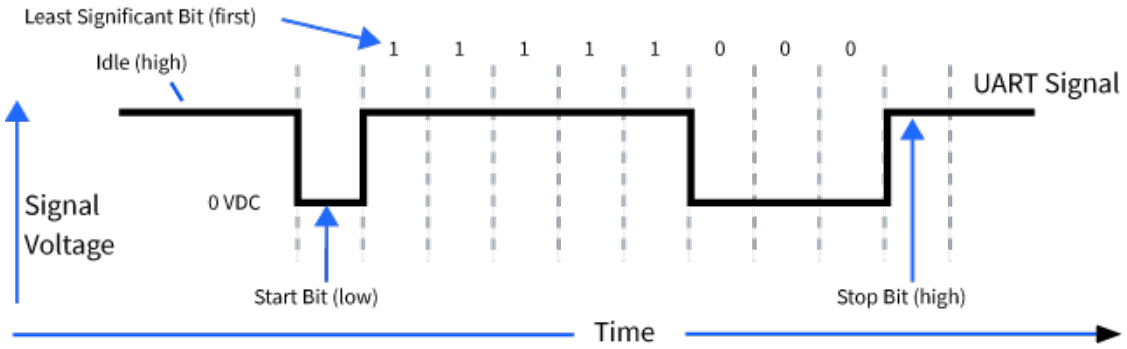
For more information about hardware specifications for the UART, see the *XBee 3 Hardware Reference Manual*.

Serial data

A device sends data to the XBee 3 Zigbee 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 Zigbee RF Module) with compatible settings for the baud rate, parity, start bits, stop bits, and data bits.

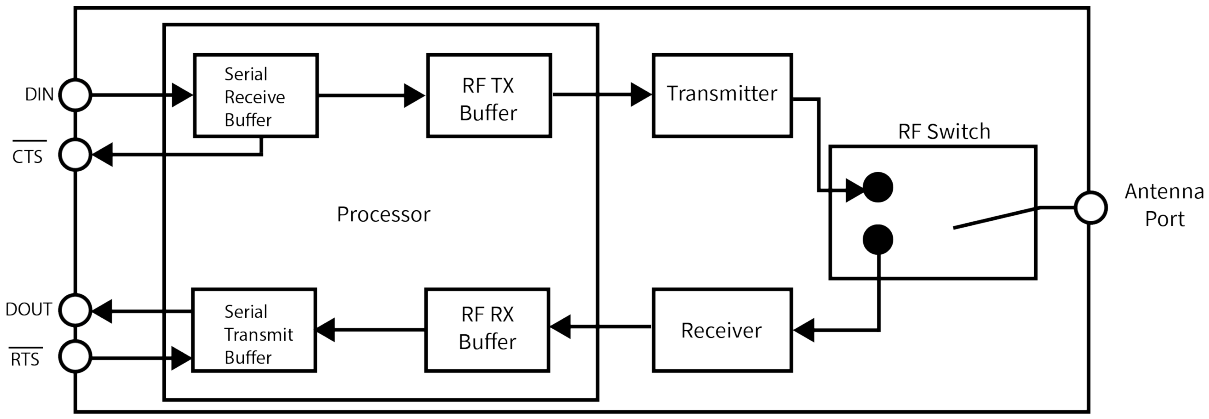
Each data byte consists of a start bit (low), 8 data bits (least significant bit first) and a stop bit (high). The following diagram illustrates the serial bit pattern of data passing through the device. The diagram shows UART data packet 0x1F (decimal number 31) as transmitted through the device.



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.

Serial buffers

The XBee 3 Zigbee RF Module maintains internal 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.



Serial receive buffer

When serial data enters the XBee 3 Zigbee RF Module through the serial port, the device stores the data in the serial receive buffer until it can be processed. Under certain conditions, the device may receive data when the serial receive buffer is already full. In that case, the device discards the data.

The serial receive buffer becomes full when data is streaming into the serial port faster than it can be processed and sent over the air (OTA). The size of the Serial receive buffer is 768 Bytes; the serial buffer may be reduced in size if RAM requirements cannot be met in future firmware releases. While the speed of receiving the data on the serial port can be much faster than the speed of transmitting data for a short period, sustained operation in that mode causes the device to drop data due to running out of places to put the data. Some things that may delay over the air transmissions are address discovery, route discovery, and retransmissions. Processing received RF data can also take away time and resources for processing incoming serial data.

If the UART is the serial port and you enable the CTS flow control, the device alerts the external data source when the receive buffer is almost full. The host delays sending data to the device until the module asserts CTS again, allowing more data to come in.

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 1056 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.

In situations where the serial transmit buffer may become full, resulting in dropped RF packets:

1. If the RF data rate is set higher than the interface data rate of the 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.

UART flow control

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

CTS flow control

If you enable CTS flow control (**D7** command), when the serial receive buffer is 17 bytes away from being full, the device de-asserts CTS (sets it high) to signal to the host device to stop sending serial data.

RTS flow control

If you set D6 (DIO6/RTS) 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.

Break control

If a serial break—DIN held low—signal is sent for over five seconds, the device resets, and it boots into Command mode with default baud settings—9600 baud. Note that after receiving the **OK** prompt, serial break must be released in order to allow input from the keyboard at 9600 baud. If either **P3** or **P4** are not enabled, this break function is disabled.

I2C

I2C master operation is supported using MicroPython.

See the *Class I2C: two-wire serial protocol* section in the *Digi MicroPython Programming Guide* for details.

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 Zigbee 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.

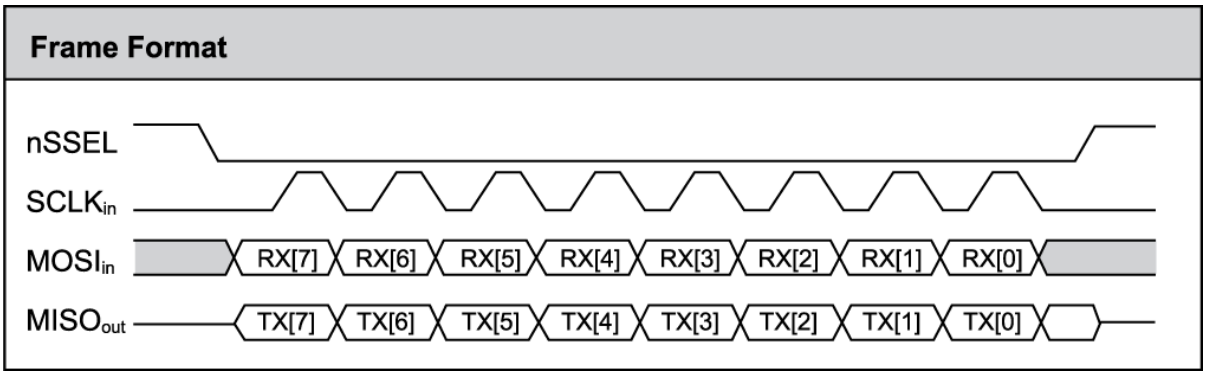
Refer to the XBee 3 Hardware Reference Guide for the pinout of your device.

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 Zigbee 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.



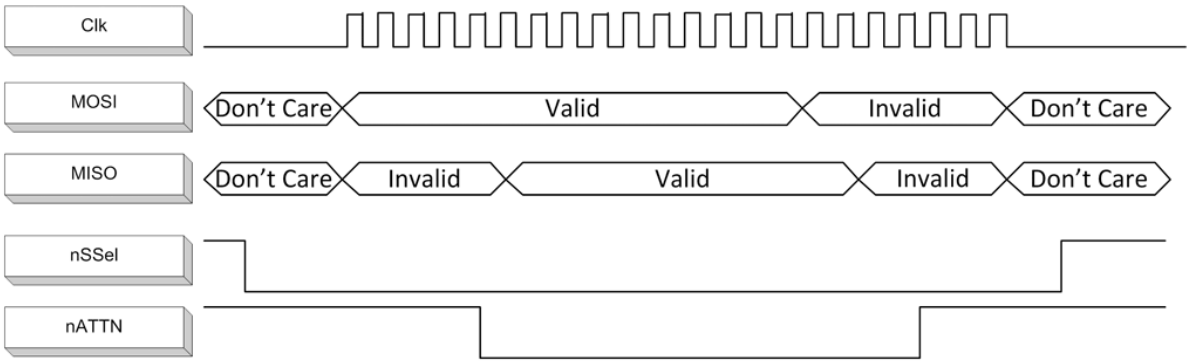
SPI mode is chip to chip communication. We do not supply a SPI communication interface on the XBee development evaluation boards included in the development kit.

Full duplex operation

When using SPI on the XBee 3 Zigbee 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 (AD1/DIO1/TH_SPI_ATTN Configuration)	P9 (DIO19/SPI_ATTN Configuration)	$\overline{\text{ATTN}}$
D2 (DIO2/AD2/TH_SPI_CLK Configuration)	P8 (DIO18/SPI_CLK Configuration)	SCLK
D3 (DIO3/AD3/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/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.

Modes

The XBee 3 Zigbee RF Module is in Receive Mode when it is not transmitting data. It shifts into the other modes of operation under the following conditions:

- Transmit mode (Serial data in the serial receive buffer is ready to be packetized)
- Command mode (Command mode sequence is issued)
- Sleep mode

Transparent operating mode

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

Serial-to-RF packetization

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

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

API operating mode

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

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

The API operation option facilitates many operations such as:

- Transmitting data to multiple destinations without entering Command mode
- Receive success/failure status of each transmitted RF packet
- Identify the source address of each received packet

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 Zigbee 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 Zigbee 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 Zigbee 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.

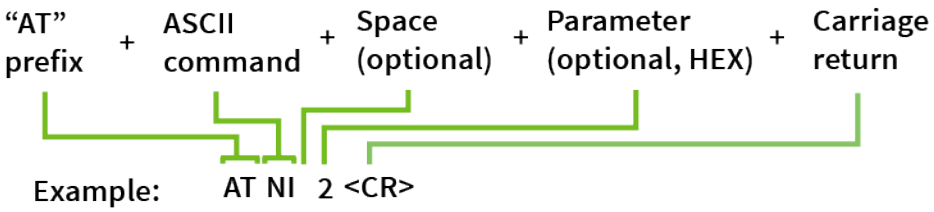
Note You must assert $\overline{\text{RTS}}$ for both of these methods, otherwise the device enters the bootloader.

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>**.

Note The behavior of the comma is the same as the behavior of the <CR> in the previous example except that the next command following the comma is not preceded by AT. The only real purpose of the comma is to reduce keystrokes.

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 Zigbee 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 device is in Idle mode. During Idle mode, the device listens for valid data on both the RF and serial ports.

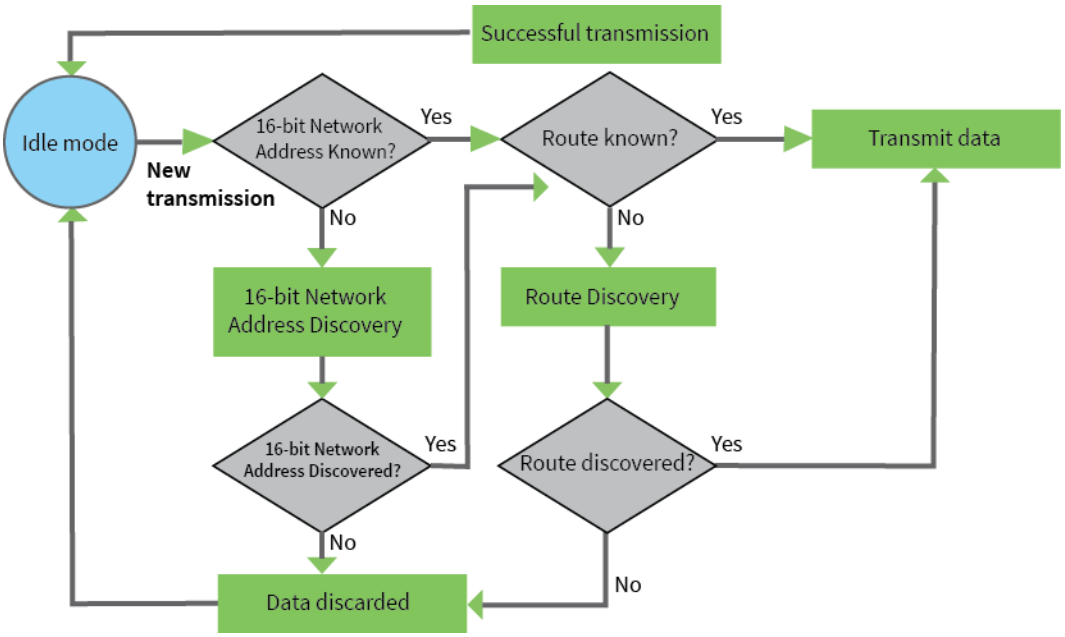
The device shifts into the other modes of operation under the following conditions:

- Transmit mode (serial data in the serial receive buffer is ready to be packetized).
- Receive mode (valid RF data received through the antenna).
- Command mode (Command mode sequence issued).

Transmit mode

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

If a 16-bit network address is not provided, a Network Address Discovery takes place. In order for data to be sent, a route discovery takes place for the purpose of establishing a route to the destination node. If a device with a matching network address is not discovered, it discards the packet. The device transmits the data once a route is established. If route discovery fails to establish a route, the device discards the packet. The following diagram shows the Transmit Mode sequence.



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

It is possible in rare circumstances for the destination to receive a data packet, but for the source to not receive the network acknowledgment. In this case, the source retransmits the data, which can cause the destination to receive the same data packet multiple times. The XBee modules do not filter out duplicate packets. We recommend that the application includes provisions to address this issue.

For more information, see Transmission, addressing, and routing.

Receive mode

When data is received over the air, the device sends the data out the serial port.

You can use the **AP** and **AO** parameters to adjust the format and types of messages that are emitted out of the serial port. Depending on your needs, you can adjust the amount of information that you receive.

By default, the device operates in Transparent mode where the the device will only output the payload of received packets. In API modes, the entire packet is emitted, and **AO** adjusts whether raw ZDO messages should be emitted.

Sleep mode

Sleep modes allow the device to enter states of low power consumption when not in use. The XBee 3 Zigbee RF Module supports both pin sleep (Sleep mode entered on pin transition) and cyclic sleep (device sleeps for a fixed time).

Sleep modes allow the device to enter states of low power consumption when not in use. The device is almost completely off during sleep, and is incapable of sending or receiving data until it wakes up. XBee devices support pin sleep, where the device enters sleep mode upon pin transition, and cyclic sleep, where the device sleeps for a fixed time.

For more information, see Manage End Devices.

Zigbee networks

The Zigbee specification

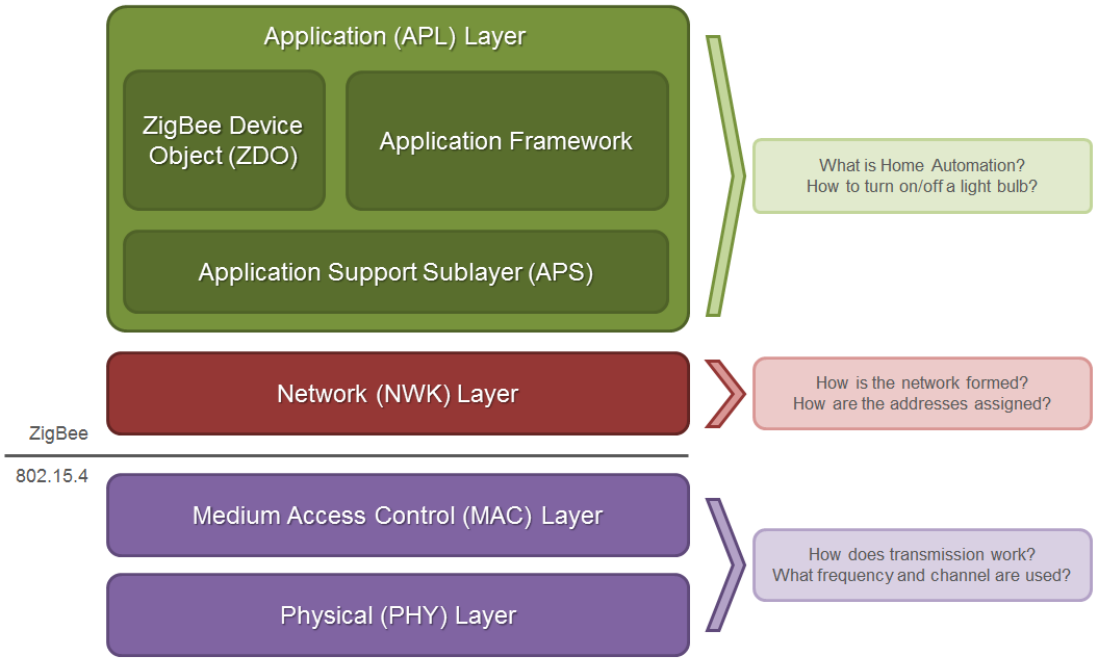
Zigbee is an open global standard for low-power, low-cost, low-data-rate, wireless mesh networking based on the IEEE 802.15.4 standard. It represents a network layer above the 802.15.4 layers to support advanced mesh routing capabilities. The Zigbee specification is developed by a consortium of companies that make up the Zigbee Alliance. For more information, see zigbee.org.

Zigbee stack layers

Most network protocols use the concept of layers to separate different components and functions into independent modules that can be assembled in different ways.

Zigbee is built on the Physical (PHY) layer and Medium Access Control (MAC) sub-layer defined in the IEEE 802.15.4 standard. These layers handle low-level network operations such as addressing and message transmission/reception.

The Zigbee specification defines the Network (NWK) layer and the framework for the application (APL) layer. The Network layer takes care of the network structure, routing, and security. The application layer framework consists of the Application Support sub-layer (APS), the Zigbee device objects (ZDO) and user-defined applications that give the device its specific functionality.



This table describes the Zigbee layers.

Zigbee layer	Descriptions
PHY	Defines the physical operation of the Zigbee device including receive sensitivity, channel rejection, output power, number of channels, chip modulation, and transmission rate specifications. Most Zigbee applications operate on the 2.4 GHz ISM band at a 250 kb/s data rate. See the IEEE 802.15.4 specification for details.

Zigbee layer	Descriptions
MAC	Manages RF data transactions between neighboring devices (point to point). The MAC includes services such as transmission retry and acknowledgment management, and collision avoidance techniques (CSMA-CA).
Network	Adds routing capabilities that allows RF data packets to traverse multiple devices (multiple hops) to route data from source to destination (peer to peer).
APS (AF)	Application layer that defines various addressing objects including profiles, clusters, and endpoints.
ZDO	Application layer that provides device and service discovery features and advanced network management capabilities.

Zigbee networking concepts

Device types

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

Coordinator

Zigbee networks may only have a single coordinator device. This device:

- Starts the network, selecting the channel and PAN ID (both 64-bit and 16-bit).
- Distributes 16-bit network addresses, allowing routers and end devices to join the network. Assists in routing data.
- Buffers wireless data packets for sleeping end device children.
- Manages the other functions that define the network, secure it, and keep it healthy.
- Cannot sleep; the coordinator must be powered on all the time.

Router

A router is a full-featured Zigbee node. This device:

- Can join existing networks and send, receive, and route information. Routing involves acting as a messenger for communications between other devices that are too far apart to convey information on their own.
- Can buffer wireless data packets for sleeping end device children. Can allow other routers and end devices to join the network.
- Cannot sleep; router(s) must be powered on all the time.
- May have multiple router devices in a network.

End device

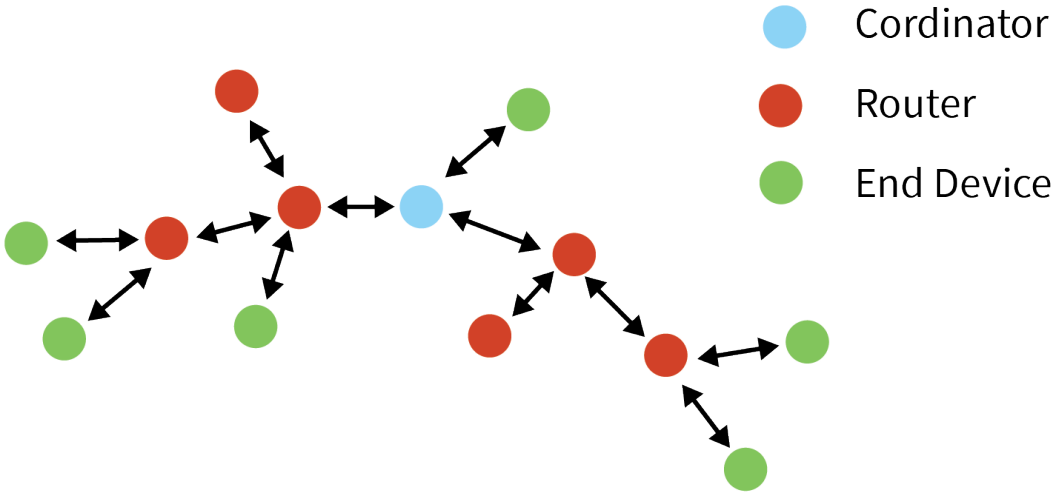
An end device is essentially a reduced version of a router. This device:

- Can join existing networks and send and receive information, but cannot act as messenger between any other devices.
- Cannot allow other devices to join the network.

- Uses less expensive hardware and can power itself down intermittently, saving energy by temporarily entering a non responsive sleep mode.
- Always needs a router or the coordinator to be its parent device. The parent helps end devices join the network, and stores messages for them when they are asleep.

Zigbee networks may have any number of end devices. In fact, a network can be composed of one coordinator, multiple end devices, and zero routers.

The following diagram shows a generic Zigbee network.



Note Each Zigbee network must be formed by one, and only one, coordinator and at least one other device (router or end device).

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

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

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

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

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

Another complication is that default parameters do not always work well for a coordinator.

For example:

- **DH**/**DL** is 0 by default, which allows routers and end devices to send transparent data to the coordinator when they first come up. If **DH**/**DL** is not changed from the default value when the device is changed to a coordinator, then the device sends data to itself, causing characters to be echoed back to the screen as they are typed. Since this is probably not the desired

operation, set **DH**/**DL** to the broadcast address or some specific unicast address when the device is changed to a coordinator.

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

PAN ID

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

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

16-bit PAN ID

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

64-bit PAN ID

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

Routers and end devices

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

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

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

Only enable **CE** on one device to avoid PAN ID conflicts and network problems.

Operating channels

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

FCC regulations mandate lower power levels on channel 26, so if you fix your network to channel 26, you will experience significantly less range on the devices.

Zigbee application layers: in depth

The following topics provide a more in-depth look at the Zigbee application stack layers (APS, ZDO) including a discussion on Zigbee endpoints, clusters, and profiles. Much of the material in these topics discuss details of the Zigbee stack that are not required in many cases.

Read these topics if:

- The XBee 3 Zigbee RF Module may talk to non-Digi Zigbee devices.
- The XBee 3 Zigbee RF Module requires network management and discovery capabilities of the ZDO layer.
- The XBee 3 Zigbee RF Module needs to operate in a public application profile (for example, smart energy, home automation, and so on).

Skip these topics if:

- The XBee 3 Zigbee RF Module does not need to interoperate or talk to non-Digi Zigbee devices.
- The XBee 3 Zigbee RF Module simply needs to send data between devices.

Application Support Sublayer (APS)

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

Application profiles

Application profiles specify various device descriptions including required functionality for various devices. The collection of device descriptions forms an application profile. Application profiles are defined as Public or Private profiles. Private profiles are defined by a manufacturer whereas public profiles are defined, developed, and maintained by the Zigbee Alliance. Each application profile has a unique profile identifier assigned by the Zigbee Alliance.

Examples of public profiles include:

- Home automation
- Smart Energy
- Commercial building automation

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

Digi XBee Zigbee firmware operates on a private profile called the Digi Drop-In Networking profile. However, in many cases the XBee 3 Zigbee RF Module can use API mode to talk to devices in public profiles or non-Digi private profiles. For more information, see API Operation.

Clusters

A cluster is an application message type defined within a profile. You can use clusters to specify a unique function, service, or action. The following examples are some clusters defined in the home automation profile:

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

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

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

XBee modules implement various clusters in the Digi private profile. You can also use the API to send or receive messages on any cluster ID (and profile ID or endpoint). For more information, see Explicit Receive Indicator - 0x91.

Endpoints

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

No TX Status frame is generated for API frames that have both **0xE6** as the destination endpoint and **0xC105** as the Profile ID as this combination is reserved for internal XBee 3 Zigbee RF Module operations.

Endpoints **0xDC** - **0xEE** are reserved for special use by Digi and should not be used in an application outside of the listed purpose. The reserved Digi endpoints are:

- 0xE8 - Digi data endpoint
- 0xE6 - Digi device object endpoint
- 0xE5 - Secure Session Server endpoint
- 0xE4 - Secure Session Client endpoint
- 0xE3 - Secure Session SRP authentication endpoint

Zigbee device profile

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

Zigbee device objects

The ZDO (endpoint 0) supports the discovery and management capabilities of the Zigbee device profile. See the Zigbee specification for a complete listing of all ZDP services. Each service has an associated cluster ID.

The XBee Zigbee firmware allows applications to easily send ZDO messages to devices in the network using the API. For more information, see ZDO transmissions.

Zigbee coordinator operation

Form a network

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

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

Security policy

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

Channel selection

When starting a network, the coordinator must select a “good” channel for the network to operate on. To do this, it performs an energy scan on multiple channels (that is, frequencies) to detect energy levels. You can use SD (Scan Duration) to adjust how long the device dwells on each channel during this energy scan. The coordinator removes channels with excessive energy levels from its list of potential channels to start on and then selects a channel at random to form the network on.

PAN ID selection

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

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

Persistent data

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

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

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

Coordinator startup

The following table provides the network formation commands that the coordinator uses to form a network.

Command	Description
CE	Must be set to 1 to specify that the device will act as a coordinator and form a network.
ID	Used to determine the 64-bit PAN ID. If set to 0 (default), a random 64-bit PAN ID will be selected.
SC	Determines the scan channels bitmask used by the coordinator when forming a network. The coordinator will perform an energy scan on all enabled **SC** channels. It will then perform a PAN ID scan.
SD	Set the scan duration, or time that the router will listen for beacons on each channel.
ZS	Set the Zigbee stack profile for the network.
EE	Enable or disable security in the network.
KY	If encryption is enabled, a preconfigured link key can be set. Any device with a matching link key will be allowed to join when the join window is open. If **KY** is set to 0, a random link key will be assigned, and devices will have to be registered to join or allowed to insecurely join using a default link key.
NK	Set a preconfigured network key for secured networks. **NK** is only applicable to the device with **CE** = **1** and defines the initial network key. In most situations you should leave this value at **0**.
EO	Set the security policy for the network if encryption is enabled. **EO** defines whether the coordinator should act as a centralized trust center or form the network as a router in a distributed trust center network. You can also optionally allow insecure devices to join using a well-known link key.

Configuration changes delay the start of network formation for five seconds after the last change.

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

When the coordinator has successfully started a network, it:

- Allows other devices to join the network for a time; see NJ (Node Join Time)
- Sets **AI** = **0**
- Starts blinking the Associate LED
- Sends an API modem status frame (“coordinator started”) out the serial port when using API mode

These behaviors are configurable using the following commands:

Command	Description
NJ	Sets the permit-join time on the coordinator, measured in seconds.
D5	Enables the Associate LED functionality.
LT	Sets the Associate LED blink time when joined. If **LT** = **0**, the default is 1 blink per 500 ms (coordinator) 250 ms (router/end device).

If any of the command values in the network formation commands table changes, the coordinator leaves its current network and starts a new network, possibly on a different channel.

Note Command changes must be applied (**AC** or **CN** command) before taking effect.

Permit joining

You can use NJ (Node Join Time) to configure the permit joining attribute on the coordinator. By default, the join window opens for 254 seconds, after which joining will not be allowed until the join window opens again.

Joining temporarily enabled

Set **NJ** < **0xFF**, to enable joining for only a number of seconds, based on the **NJ** parameter. Once the XBee 3 Zigbee RF Module joins a network, the timer starts. The coordinator does not re-enable joining if the device is power cycled or reset. The following actions restart the permit-joining timer:

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

The last two actions enable joining for one minute if **NJ** is **0x0**. Otherwise, the Commissioning button and the CB2 command enable joining for **NJ** seconds.

Joining always enabled

If **NJ** = **0xFF**, joining is permanently enabled.

Use this mode carefully. Once a network has been deployed, we strongly recommend that the application consider disabling joining to prevent unwanted joins from occurring. An always-open network operates outside of the Zigbee 3.0 specifications.

Reset the coordinator

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

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

Leave a network

The following mechanisms cause the coordinator to leave its current PAN and start a new network based on its network formation parameter values.

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

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

Replace a coordinator (security disabled only)

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

Note Avoid having two coordinators on the same channel, stack profile, and PAN ID (16-bit and 64-bit) as it can cause problems in the network. When replacing a coordinator, turn off the old coordinator before starting the new coordinator.

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

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

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

Command	Description
ID	Set the 64-bit PAN ID to match the read **OP** value.

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

II is the initial 16-bit PAN ID. Under certain conditions, the Zigbee stack can change the 16-bit PAN ID of the network. For this reason, you cannot save the **II** command using the **WR** command. Once **II** is set, the coordinator leaves the network and starts on the 16-bit PAN ID specified by **II**.

Example: start a coordinator

1. Set CE (Device Role) to **1** to indicate to the local device that it should form a network. Use WR (Write) to save the changes.
2. Set **SC** and **ID** to the desired scan channels and PAN ID values. The defaults are usually sufficient.
3. If you change **SC** or **ID** from the default, issue the **WR** command to save the changes.
4. If you change **SC** or **ID** from the default, apply changes (make **SC** and **ID** changes take effect) either by sending the **AC** command or by exiting **AT** Command mode.
5. If an Associate LED has been connected, it starts blinking once the coordinator has selected a channel and PAN ID and the network has started.
6. The API Modem Status frame (Coordinator Started) is sent out the serial port when using API mode.
7. Reading the **AI** command (association status) returns a value of 0, indicating a successful startup.
8. Reading the **MY** command (16-bit address) returns a value of 0, the Zigbee-defined 16-bit address of the coordinator.

After startup, the coordinator allows joining based on its **NJ** value. We highly recommend that you issue a **WR** command to write all applied settings to flash.

Example: replace a coordinator (security disabled)

1. Read the **OP**, **OI**, **CH**, and **ZS** commands on the running coordinator.
2. Set the **CE**, **ID**, **SC**, and **ZS** parameters on the new coordinator to match the existing coordinator, followed by **WR** command to save these parameter values.
3. Turn off the running coordinator.
4. Set the II (Initial 16-bit PAN ID) parameter on the new coordinator to match the read **OI** value on the old coordinator.
5. Wait for the new coordinator to start (**AI** = **0**).

Router operation

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

Discover Zigbee networks

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

The beacon contains information about the PAN the nearby device is on, including the PAN identifier (PAN ID), and whether or not joining is allowed. The router evaluates each beacon received on the channel to determine if it finds a valid PAN. A PAN is valid if any of the following exist:

- Has a valid 64-bit PAN ID (PAN ID matches **ID** if **ID** > 0)
- Has the correct stack profile (**ZS** command)
- Allows joining the network

If the router does not find a valid PAN, it performs the PAN scan on the next channel in its scan channels list and continues scanning until it finds a valid network, or until all channels have been scanned. If the rounter scans all channels and does not discover a valid PAN, it scans all channels again.

The Zigbee Alliance requires that certified solutions not send beacon request messages too frequently. To meet certification requirements, the XBee firmware attempts nine scans per minute for the first five minutes, and three scans per minute thereafter. If a valid PAN is within range of a joining router, it typically discovers the PAN within a few seconds.

Join a network

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

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

Authentication

In a network where security is enabled, the router must follow an authentication process. See Zigbee security for a discussion on security and authentication.

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

Persistent data

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

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

The router retains this information indefinitely until it leaves the network. When the router leaves a network, it loses the previous PAN ID, operating channel, and child table data.

Router joining

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

Set DJ (Disable Joining) to **1** to disable joining. You cannot write the **DJ** parameter with the **WR** command, so a power cycle always clears the **DJ** setting.

The following commands control the router joining process.

Command	Description
ID	Sets the 64-bit PAN ID to join. Setting **ID** = **0** allows the router to join any 64-bit PAN ID.
SC	Set the scan channels bitmask that determines which channels a router scans to find a valid network. Set **SC** on the router to match **SC** on the coordinator. For example, setting **SC** to 0x281 enables scanning on channels 11, 18 and 20, in that order.
SD	Set the scan duration, or time that the router listens for beacons on each channel.
ZS	Set the stack profile on the device.
EE	Enable or disable security in the network. This must be set to match the **EE** value (security policy) of the coordinator.
KY	Set the trust center link key. If set to 0 (default), the link key is expected to be obtained (unencrypted) during joining.
EO	If encryption is enabled (**EE** = **1**), set the joining device's Encryption Options to match the Encryption Options of the network.

Configuration changes delay the start of joining for five seconds after the last change.

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

If any of the above command values change, when command register changes are applied (**AC** or **CN** commands), the router leaves its current network and attempts to discover and join a new valid network. When a Zigbee router has successfully joined a network, it:

- Allows other devices to join the network for a time
- Sets **AI** = **0**
- Starts blinking the Associate LED

- Sends an API modem status frame (associated) out the serial port when using API mode

You can configure these behaviors using the following commands:

Command	Description
NJ	Sets the permit-join time on the router, or the time that it allows new devices to join the network, measured in seconds. Set **NJ** = **0xFF** to always enable permit joining.
D5	Enables the Associate LED functionality.
LT	Sets the Associate LED blink time when joined. The default is 2 blinks per second (router).

Router network connectivity

Once a router joins a Zigbee network, it remains connected to the network on the same channel and PAN ID unless it is forced to leave (see Leave a network). If the scan channels (**SC**), PAN ID (**ID**) and security settings (**EE**, **KY**) do not change after a power cycle, the router remains connected to the network after a power cycle.

If a router is physically moved out of range of the network it initially joined, make sure the application includes provisions to detect if the router can still communicate with the original network. If communication with the original network is lost, the application may choose to force the router to leave the network. The XBee firmware includes two provisions to automatically detect the presence of a network and leave if the check fails.

Power-On join verification

JV (Coordinator Join Verification) enables the power-on join verification check. If enabled, the XBee 3 Zigbee RF Module attempts to discover the 64-bit address of the coordinator when it first joins a network. Once it has joined, it also attempts to discover the 64-bit address of the coordinator after a power cycle event. If 3 discovery attempts fail, the router leaves the network and try to join a new network. The default setting for Power-on join verification is disabled (**JV** defaults to **0**).

Network watchdog

The NW (Network Watchdog Timeout) feature allows a powered router to verify the presence of a coordinator if no communication events with the coordinator have occurred within a timeout period. This timeout is specified in minutes using the **NW** command. The default setting for the network watchdog feature is disabled (**NW** = **0**) and can be configured for up to several days.

Anytime a router receives valid data from the coordinator or data collector, it clears the watchdog timeouts counter and restarts the watchdog timer.

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

If any of the events listed above occur during the watchdog period then no additional network traffic will be generated. If the watchdog timer does expire (no valid data received for 1 **NW** time period), the router attempts to initiate communication with the coordinator by sending an IEEE 64-bit address discovery message to the coordinator. If the router cannot discover the address, it records one watchdog timeout. After three consecutive network watchdog timeouts expire (3 * **NW**) and the coordinator has not responded to the address discovery attempts, the router will enter one of three

modes based on the configuration of DC (Joining Device Controls) bit 5 and DO (Miscellaneous Device Options) bit 7:

1. **No Network Locator (Leave Network)**: If neither **DC** bit 5 or **DO** bit 7 is set, the router will immediately leave the network and begin searching for a network to join based on its networking settings. If API mode is enabled, a network disassociated modem status frame (0x03) will be emitted when the router leaves the network. If the router finds and joins a new coordinator or the original coordinator, a joined network modem status frame (0x02) will be emitted if API mode is enabled.
2. **Network Locator with Network Leave**: If **DO** bit 7 is set but **DC** bit 5 is not set, the behavior of **JV** and **NW** are modified. The router will remain on the network until a new network is found. The router starts scanning for a network across the channels of the Scan Channel mask (SC). Scanning occurs at a random interval of between 90 and 135 seconds. If API mode is enabled, a network watchdog scanning modem status frame (0x42) will be emitted when scanning begins. If the device finds a network on the old channel with the same **OI** and operating **ID**, the search mode ends. If the device finds a network with a new **OI** but satisfies the device's search for a matching **ID** and **ZS**, the device leaves the old network and joins the new network with the new **OI**. These leave and join actions will cause the router to emit a disassociated network modem status (0x03) and a joined network modem status frame (0x02) if API mode is enabled. This supports swapping or replacing a coordinator in a running network.
3. **Network Locator with Rejoin**: If **DC** bit 5 is set, the router will begin scanning the current channel indefinitely in an attempt to find the coordinator on the original network or re-join the coordinator if it has moved to a new network. If API mode is enabled, a network watchdog scanning modem status frame (0x42) will be emitted when scanning begins. Scanning occurs at a random interval of between 90 and 135 seconds. If it finds the coordinator which must have a matching **ID** (extended PAN ID) with the same PAN ID (**OI**) or a new PAN ID, the router will rejoin the coordinator even if the coordinator is configured with a **NJ** of **0** (joining disabled). If the router finds the coordinator on the original network or rejoins coordinator on a new network, a joined network modem status frame (0x02) will be emitted if API mode is enabled.

Leave a network

The following mechanisms cause the coordinator to leave its current PAN and start a new network based on its network formation parameter values.

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

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

Reset the router

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

To prevent the router from leaving an existing network, issue the **WR** command after all network joining commands have been configured; this retains the settings through power cycle or reset events.

Example: join a network

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

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

End device operation

Similar to routers, end devices must discover and join a valid Zigbee network before they can participate in the network. After an end device joins a network, it can communicate with other devices on the network. Because end devices are battery powered and support low power (sleep) modes, they cannot allow other devices to join or route data packets.

Discover Zigbee networks

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

- Has a valid 64-bit PAN ID (PAN ID matches ID if **ID** > **0**)
- Has the correct stack profile (**ZS** command)
- Allows joining the network
- Has capacity for additional end devices

If the end device does not find a valid PAN, it performs the PAN scan on the next channel in its scan channels list and continues this process until it finds a valid network, or until all channels have been

scanned. If the end device scan all channels and does not discover a valid PAN, it may enter a low power sleep state and scan again later.

If scanning all **SC** channels fails to discover a valid PAN, XBee Zigbee devices attempt to enter a low power state and retries scanning all **SC** channels after the device wakes from sleeping. If the device cannot enter a low power state, it retries scanning all channels, similar to the router. To meet Zigbee Alliance requirements, the end device attempts up to nine scans per minute for the first five minutes, and three scans per minute thereafter.

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

Join a network

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

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

Parent child relationship

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

End device capacity

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

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

In Zigbee firmware, use the **NC** command (number of remaining end device children) to determine how many additional end devices can join a router or coordinator. If **NC** returns 0, then the router or coordinator device has no more end device capacity.

Note Because routers cannot sleep, there is no equivalent need for routers or coordinators to track joined routers. There is no limit to the number of routers that can join a given router or coordinator device and no “router capacity” metric.

Authentication

In a network where security is enabled, the end device must then go through an authentication process. For more information, see Zigbee security.

Persistent data

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

Orphan scans

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

If the orphaned end device receives a coordinator realignment command, it joins the network. Otherwise, it attempts to discover and join a valid network.

End device joining

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

Note Set the **DJ** command to 1 to disable joining. You cannot write the **DJ** parameter with **WR**, so a power cycle always clears the **DJ** setting.

The following commands control the end device joining process.

Command	Description
ID	Sets the 64-bit PAN ID to join. Setting **ID** = **0** allows the router to join any 64-bit PAN ID.
SC	Set the scan channels bitmask that determines which channels an end device will scan to find a valid network. **SC** on the end device should be set to match SC on the coordinator and routers in the desired network. For example, setting **SC** to **0x281** enables scanning on channels 0x0B, 0x12, and 0x14, in that order.
SD	Set the scan duration, or time that the end device will listen for beacons on each channel.
ZS	Set the stack profile on the device.
EE	Enable or disable security in the network. This must be set to match the **EE** value (security policy) of the coordinator.
KY	Set the trust center link key. If set to 0 (default), the link key is expected to be obtained (unencrypted) during joining.
EO	If encryption is enabled (**EE** = **1**), set the joining device's Encryption Options to match the Encryption Options of the network.

Once the end device joins a network, the network configuration settings persist through power cycles as mentioned in Persistent data. If joining fails, read the status of the last join attempt in the **AI** command register.

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

When a Zigbee end device has successfully started a network, it:

- Sets **AI** equal to **0**
- Starts blinking the Associate LED if one has been connected to the device's ASSC pin (Micro pin 26/SMT pin 28/TH pin 15)
- Sends an API modem status frame ("associated") out the serial port when using API mode
- Attempts to enter the sleep mode defined by the **SM** parameter

You can use the following commands to configure these behaviors:

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

Parent connectivity

The XBee 3 Zigbee RF Module end device sends regular poll transmissions to its parent when it is awake. These poll transmissions query the parent for any new received data packets. The parent always sends a MAC layer acknowledgment back to the end device. The acknowledgment indicates whether the parent has data for the end device.

If the end device does not receive an acknowledgment for three consecutive poll requests, it considers itself disconnected from its parent and attempts to discover and join a valid Zigbee network. For more information, see Manage End Devices.

Reset the end device

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

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

Channel scanning

Routers and end devices must scan one or more channels to discover a valid network to join. When a join attempt begins, the device sends a beacon request transmission on the lowest channel specified

in the SC (Scan Channels) bitmask. If the device finds a valid PAN on the channel, it attempts to join the PAN on that channel. Otherwise, if the device does not find a valid PAN on the channel, it attempts scanning on the next higher channel in the **SC** bitmask.

The device continues to scan each channel (from lowest to highest) in the **SC** bitmask until it finds a valid PAN or all channels have been scanned. Once the device scans all channels, the next join attempt starts scanning on the lowest channel specified in the **SC** bitmask.

For example, if the **SC** command is set to **0x400F**, the device starts scanning on channel 11 (**0x0B**) and scans until it finds a valid beacon, or until it scans channels 11, 12, 13, 14, and 25 have been scanned (in that order).

Once an XBee router or end device joins a network on a given channel, if the XBee device receives a network leave command (see Leave a network), it leaves the channel it joined on and continues scanning on the next higher channel in the **SC** bitmask.

For example, if the **SC** command is set to **0x400F** and the device joins a PAN on channel 12 (**0x0C**), if the XBee 3 Zigbee RF Module leaves the channel, it starts scanning on channel 13, followed by channels 14 and 25 if it does not find a valid network. Once all channels have been scanned, the next join attempt starts scanning on the lowest channel specified in the **SC** bitmask.

Manage multiple Zigbee networks

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

- PAN ID filtering
- Preconfigured security keys
- Permit joining
- Application messaging

Filter PAN ID

Set ID (Extended PAN ID) to a non-zero value to configure the XBee 3 Zigbee RF Module with a fixed PAN ID.

If you set the PAN ID to a non-zero value, the device will only join a network with the same PAN ID.

Configure security keys

Similar to PAN ID filtering, this method requires that you install a known security key on a router to ensure it joins a Zigbee network with the same security key.

1. Use EE (Encryption Enable) to enable security.
2. Use KY (AES Encryption Key) to set the preconfigured link key to a non-zero value.

Now the XBee router or end device will only join a network with the same security key.

Prevent unwanted devices from joining

You can disable the permit-joining parameter in a network to prevent unwanted devices from joining. When you need to add a new device to a network, enable permit-joining for a short time on the desired network.

In the XBee firmware:

1. Set NJ (Node Join Time) to a value less than **0xFF** on all routers and coordinator devices to restrict joining (recommended).
2. Use the Commissioning pushbutton or CB (Commissioning Pushbutton) to allow joining for a short time; for more information, see Network commissioning and diagnostics.

Application messaging framework

If none of the previous mechanisms are feasible, you can build a messaging framework between the coordinator and devices that join its network into the application. For example, the application code in joining devices could send a transmission to the coordinator after joining a network, and wait to receive a defined reply message. If the application does not receive the expected response message after joining, it could force the device to leave and continue scanning; see NR (Network Reset).

Transmission, addressing, and routing

Addressing

All Zigbee devices have two different addresses, a 64-bit and a 16-bit address. This section describes the characteristics of each.

64-bit device addresses

The 64-bit address is a device address which is unique to each physical device. It is sometimes also called the MAC address or extended address and is assigned during the manufacturing process. The first three bytes of the 64-bit address is a Organizationally Unique Identifier (OUI) assigned to the manufacturer by the IEEE. The OUI of XBee devices is 0x0013A2.

16-bit device addresses

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

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

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

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

Application layer addressing

Zigbee devices support multiple application profiles, cluster IDs, and endpoints (for more information, see Zigbee application layers: in depth). Application layer addressing allows data transmissions to be addressed to specific profile IDs, cluster IDs, and endpoints. Application layer addressing is useful if an application must do any of the following:

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

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

Data transmission

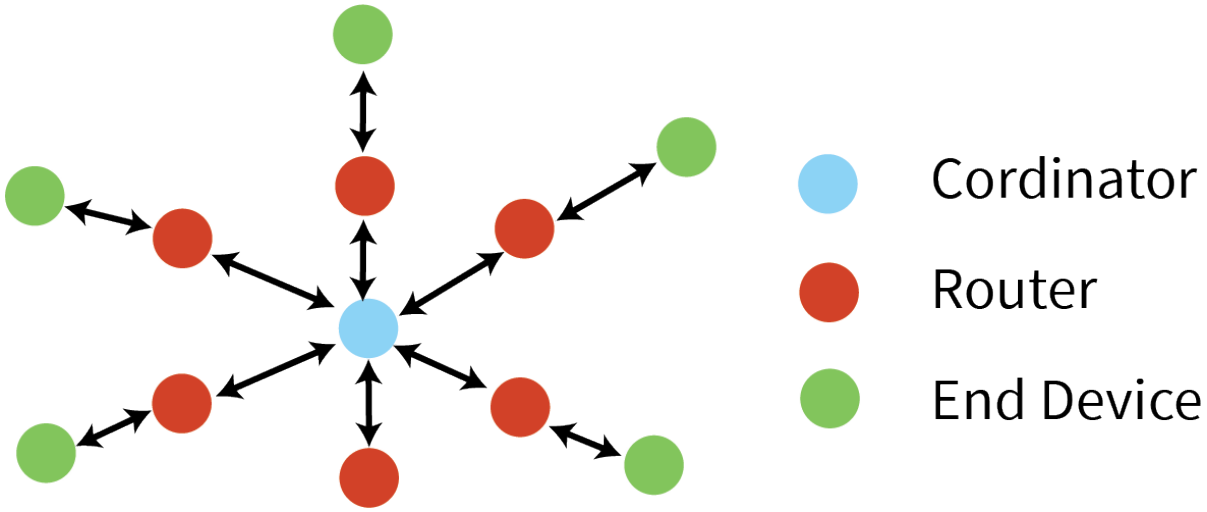
You can send Zigbee data packets as either unicast or broadcast transmissions. Unicast transmissions route data from one source device to one destination device, whereas broadcast transmissions are sent to many or all devices in the network.

Broadcast transmissions

Broadcast transmissions within the Zigbee protocol are intended to be propagated throughout the entire network such that all nodes receive the transmission. To accomplish this, the coordinator and

all routers that receive a broadcast transmission retransmits the packet three times.

Note When a router or coordinator delivers a broadcast transmission to an end device child, the transmission is only sent once (immediately after the end device wakes and polls the parent for any new data). For more information, see Parent operation.



Each node that transmits the broadcast also creates an entry in a local broadcast transmission table. This entry keeps track of each received broadcast packet to ensure the packets are not transmitted endlessly. Each entry persists for 8 seconds, and the broadcast transmission table holds 8 entries, effectively limiting network broadcast transmissions to once per second.

For each broadcast transmission, the Zigbee stack reserves buffer space for a copy of the data packet that retransmits the packet as needed. Large broadcast packets require more buffer space. Users cannot change any buffer spacing; information on buffer space is for general knowledge only. The XBee 3 Zigbee RF Module handles buffer spacing automatically.

Note Broadcast transmissions do not use ACKs, so there is no guarantee that every node will hear a particular broadcast. Because the XBee devices re-transmit broadcast transmissions by every device in the network, use broadcast messages sparingly.

Unicast transmissions

Unicast transmissions are sent from one source device to another destination device. The destination device could be an immediate neighbor of the source, or it could be several hops away. Unicast transmissions sent along a multiple hop path require some means of establishing a route to the destination device. For more information, see RF packet routing.

Address resolution

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

When sending a unicast transmission, the Zigbee network layer uses the 16-bit address of the destination and each hop to route the data packet. If you do not know the 16-bit address of the destination, the Zigbee stack includes a discovery provision to automatically discover the destination 16-bit address of the device before routing the data.

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

You can address frames using either the extended or the network address. If you use the extended address form, set the 16-bit network address field to 0xFFFE (unknown). If you use the 16-bit network address form, set the 64-bit extended address field to 0xFFFFFFFFFFFFFFFF (unknown).

If you use an invalid 16-bit address as a destination address, and the 64-bit address is unknown (0xFFFFFFFFFFFFFFFF), the modem status message shows a delivery status code of 0x21 (network ack failure) and a discovery status of 0x00 (no discovery overhead). If you use a non-existent 64-bit address as a destination address, and the 16-bit address is unknown (0xFFFE), the device attempts address discovery and the modem status message shows a delivery status code of 0x24 (address not found) and a discovery status code of 0x01 (address discovery was attempted).

Address table

Each Zigbee device maintains an address table that maps a 64-bit address to a 16-bit address. When a transmission is addressed to a 64-bit address, the Zigbee stack searches the address table for an entry with a matching 64-bit address to determining the destination's 16-bit address. If the Zigbee stack does not find a known 16-bit address, it performs address discovery to discover the device's current 16-bit address.

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

The XBee 3 Zigbee RF Module supports up to 20 address table entries. For applications where a single device (for example, coordinator) sends unicast transmissions to more than 10 devices, the application implements an address table to store the 16-bit and 64-bit addresses for each remote device. Use API mode for any XBee device that sends data to more than 10 remotes. The application can then send both the 16-bit and 64-bit addresses to the XBee device in the API transmit frames which significantly reduces the number of 16-bit address discoveries and greatly improves data throughput.

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

When sending a transmission to a destination 64-bit address, the application searches the address table for a matching 64-bit address. If it finds a match, the application populates the 16-bit address into the 16-bit address field of the API frame. If it does not find a match, set the 16-bit address to 0xFFFE (unknown) in the API transmit frame. The API provides indication of a remote device's 16-bit address in the following frames:

- All receive data frames
- Rx Data (0x90)

- Rx Explicit Data (0x91)
- I/O Sample Data (0x92)
- Node Identification Indicator (0x95)
- Route Record Indicator (0xA1) and so forth
- Transmit status frame (0x8B)

Group table

Each router and the coordinator maintain a persistent group table. Each entry contains the following:

- Endpoint value
- Two byte group ID
- Optional name string of zero to 16 ASCII characters
- Index into the binding table

More than one endpoint may be associated with a group ID, and more than one group ID may be associated with a given endpoint. The capacity of the group table is 16 entries.

The application always updates the 16-bit address in the address table when it receives one of the frames to ensure the table has the most recently known 16-bit address. If a transmission failure occurs, the application sets the 16-bit address in the table to 0xFFFE (unknown).

Binding transmissions

Binding transmissions use indirect addressing to send one or more messages to other destination devices. The device handles an Explicit Addressing Command Request - 0x11 using the Indirect Tx Option (0x04) as a binding transmission request.

Multicast transmissions

XBee modules use multicast transmissions to broadcast a message to destination devices that have active endpoints associated with a common group ID. The device handles an Explicit Addressing Command Request - 0x11 using the Multicast Tx Option (0x08) as a multicast transmission request.

Address resolution

The 64 bit destination address value does not matter and we recommend that it be set to 0xFFFFFFFFFFFFFFFF. Set the 16 bit destination address value to the destination groupID.

Address resolution

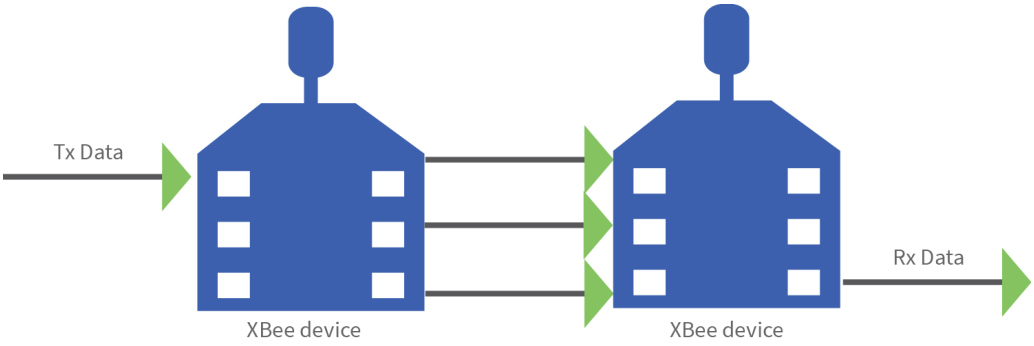
The XBee 3 Zigbee RF Module use the source endpoint and cluster ID values of a binding transmission as keys to lookup matching binding table entries. For each matching binding table entry, the type field of the entry indicates whether to send a unicast or a multicast message. In the case of a unicast entry, the transmission request is updated with the Destination Endpoint and MAC Address, and unicast to its destination. In the case of a multicast entry, the device updates the message using the two least significant bytes of the Destination MAC Address as the groupID, and multicast to its destinations.

Binding table

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

Fragmentation

Each unicast transmission may support up to 84 bytes of RF payload, although enabling security or using source routing can reduce this number. For more information, see NP (Maximum Packet Payload Bytes). However, the XBee Zigbee firmware supports a Zigbee feature called fragmentation that allows a single large data packet to be broken up into multiple RF transmissions and reassembled by the receiver before sending data out its serial port.



The transmit frame can include up to 255 bytes of data broken up into multiple transmissions and reassembled on the receiving side. If one or more of the fragmented messages are not received by the receiving device, it drops the entire message, and the sender indicates a transmission failure in Extended Transmit Status - 0x8B.

Applications that do not wish to use fragmentation should avoid sending more than the maximum number of bytes in a single RF transmission—see Maximum RF payload size.

If you use the **D6** command to enable RTS flow control on the receiving device it receives a fragmented message; it ignores RTS flow control.

Note Broadcast transmissions do not support fragmentation. Maximum payload size = up to 92 bytes.

Data transmission examples

This section provides examples for data transmission.

Send a packet in Transparent mode

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

Example: Send a transmission to the coordinator.

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

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

1. Enter Command mode (**+++**).
2. After receiving an **OK\r**, issue the following commands:
   - **ATDH0**\r
   - **ATDL0**\r
   - **ATCN**\r
3. Verify that each of the three commands returned an **OK\r** response.
4. After setting these command values, all serial characters received on the UART are sent as a unicast transmission to the coordinator.

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

1. Enter Command mode (**+++**)
2. After receiving an **OK\r**, issue the following commands:
   a. **ATDH13A200**\r
   b. **ATDL404A2244**\r
   c. **ATCN**\r
3. Verify that each of the three commands returned an **OK\r** response.
4. After setting these command values, all serial characters received on the UART are sent as a unicast transmission to the coordinator.

Send data in API mode

API mode is used exclusively for outgoing and incoming messages when the **AP** parameter is non-zero. Use the transmit request, or explicit transmit request frame (0x10 and 0x11 respectively) to send data to the coordinator. The 64-bit address can either be set to 0x0000000000000000, or to the 64-bit address of the coordinator. The 16-bit address should be set to 0xFFFE when using the 64-bit address of all 0x00s.

To send an ASCII **1** to the coordinator's 0x00 address, use the following API frame:

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

If you use the explicit transmit frame, set the the cluster ID to 0x0011, the profile ID to 0xC105, and the source and destination endpoints to 0xE8. These are the recommended defaults for data transmissions in the Digi profile.

You can send the same transmission using the following explicit transmit frame:

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

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

Suppose the coordinator's 64-bit address is 0x0013A200404A2244. The following transmit request API frame (0x10) sends an ASCII **1** to the coordinator:

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

Example: Send a broadcast transmission

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

Perform the following steps to configure a broadcast transmission:

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

API frame examples

A transmit request API frame (0x10) can send an ASCII **1** in a broadcast transmission using the following API frame:

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

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

Example: Send an indirect (binding) transmission.

This example uses the explicit transmit request frame (0x11) to send a transmission using indirect addressing through the binding table. It assumes the binding table has already been set up to map a source endpoint of **D5** and cluster ID of 0x0001 to a destination endpoint and 64 bit destination address. The message data is a manufacturing specific profile message using profile ID 0xC105, command ID 0x00, a ZCL Header of 151E10, transaction number EE, and a ZCL payload of 000102030405:

7E 00 1E 11 01 FF FF FF FF FF FF FF FF FF D5 D5 00 01 C1 05 00 04 15 1E 10 EE 00 01 02 03 04 05 42

Note The 64 bit destination address has been set to all 0xFF values, and the destination endpoint set to 0xFF. The Tx Option 0x04 indicates indirect addressing. The 64 bit destination address and destination endpoint are completed by looking up data associated with binding table entries. This matches the following example.

Example: Send a multicast (group ID) broadcast.

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

7E 00 1E 11 01 FF FF FF FF FF FF FF 12 34 D5 D5 00 01 C1 05 00 08 15 1E 10 EE 00 01 02 03 04 05 F6

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

RF packet routing

Unicast transmissions may require some type of routing. Zigbee includes several different methods to route data, each with its own advantages and disadvantages as summarized in the following table.

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

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

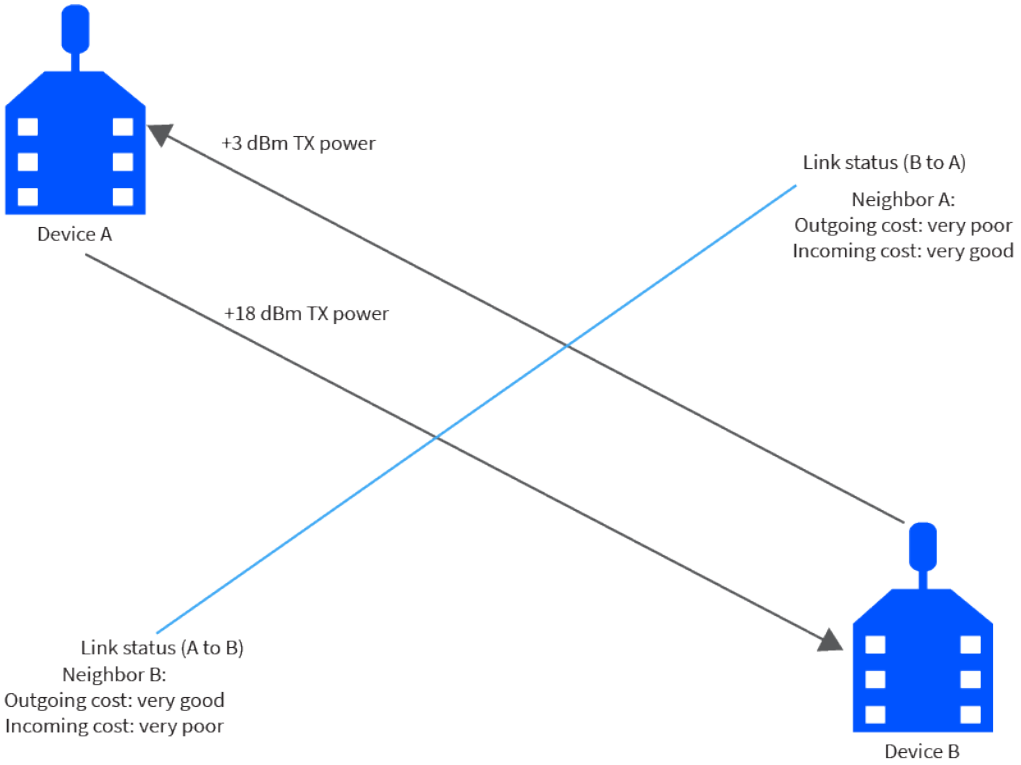
Note To revert from Many-to-One routing to AODV routing, a network must first do a network reset (**NR**).

Link status transmission

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

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

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



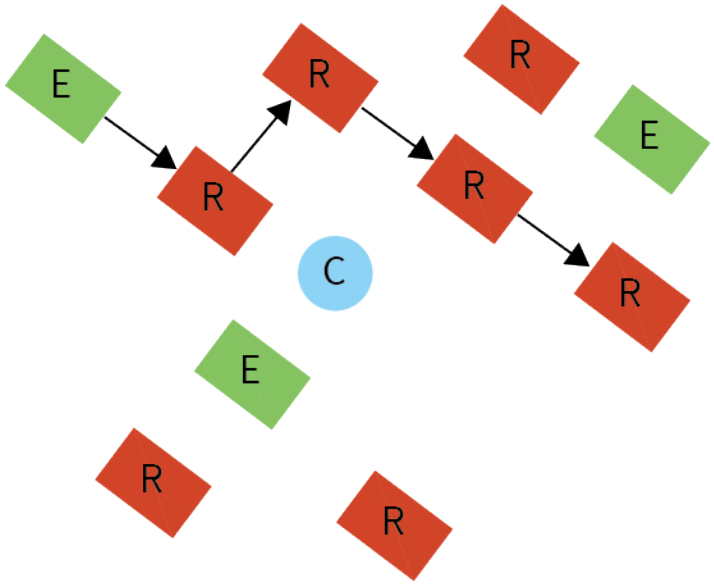
This mechanism enables devices A and B to recognize that the link is not reliable in both directions and select a different neighbor when establishing routes. Such links are called asymmetric links, meaning the link quality is not similar in both directions.

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

AODV mesh routing

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

Sample transmission through a mesh network:



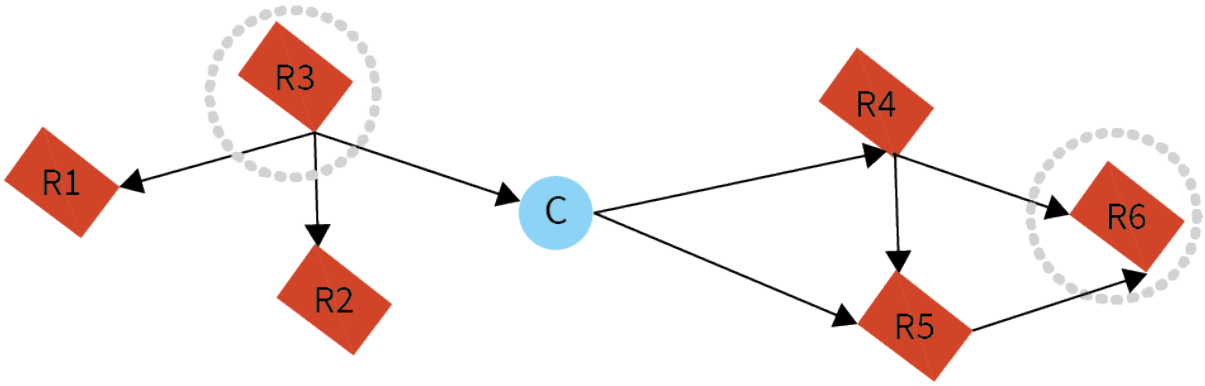
AODV routing algorithm

Routing under the AODV protocol uses tables in each node that store the next hop (intermediary node between source and destination nodes) for a destination node. If a next hop is unknown, route discovery takes place to find a path. Since only a limited number of routes can be stored on a router, route discovery takes place more often on a large network with communication between many different nodes.

Node	Destination address	Next hop address
R3	Router 6	Coordinator
C	Router 6	Router 5
R5	Router 6	Router 6

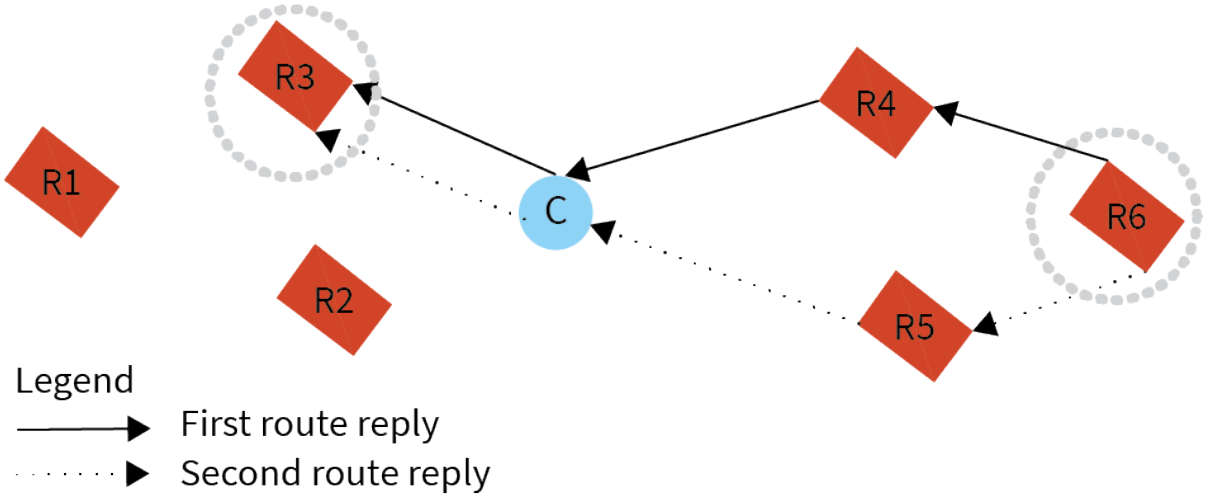
When a source node discovers a route to a destination node, it sends a broadcast route request command. The route request command contains the source network address, the destination network address and a path cost field (a metric for measuring route quality). As the route request command propagates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

The following graphic is a sample route request (broadcast) transmission where R3 is trying to discover a route to R6:



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

The following graphic is a sample route reply (unicast) where R6 sends a route reply to R3:



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

Retries and acknowledgments

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

This Ack is called the Mac layer acknowledgment. In addition, the device that originated the transmission expects to receive an acknowledgment packet (Ack) from the destination device. This Ack traverses the same path the data traversed, but in the opposite direction. If the originator fails to receive this Ack, it retransmits the data, up to two times until it receives an Ack. This Ack is called the Zigbee APS layer acknowledgment.

Note Refer to the Zigbee specification for more details.

Many-to-One routing

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

Many-to-one routing is an optimization for these kinds of networks. Rather than require each device to do its own route discovery, the device sends a single many-to-one broadcast transmission from the data collector to establish reverse routes on all devices.

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

When a device sends data to a data collector, and it finds a many-to-one route in its routing table, it transmits the data without performing a route discovery. Send the many-to-one route request periodically to update and refresh the reverse routes in the network.

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

The Zigbee firmware uses AR (Aggregate Routing Notification) to enable many-to-one broadcasting on a device. **AR** sets a time interval (measured in 10 second units) for sending the many to one broadcast transmission.

High/Low RAM Concentrator mode

When Many to One (MTO) requests are broadcast, **DO** = 40 (bit 6) determines if the concentrator is operating in high or low RAM mode. High RAM mode indicates to the network that the concentrator has enough memory to store source routes for the whole network, and remote nodes may stop sending route records after the concentrator has successfully received one. Low RAM mode indicates to the network that the concentrator lacks RAM to store route records, and that route records be sent to the concentrator to precede every inbound APS unicast message. If you have a network with more than forty devices or will be using a Digi gateway, we recommend operating in low RAM concentrator mode and externally manage source routing.

A device will become a concentrator when **AR** < **0xFF** or when acting as a Centralized Trust Center.

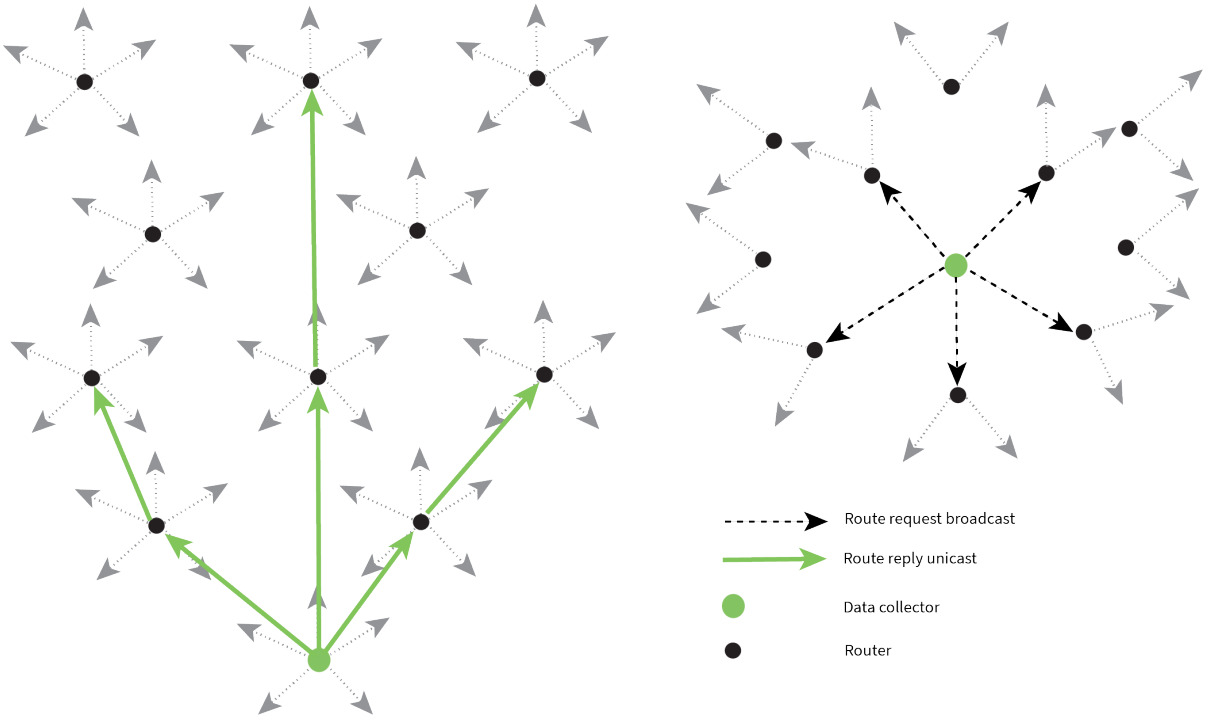
Source routing

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

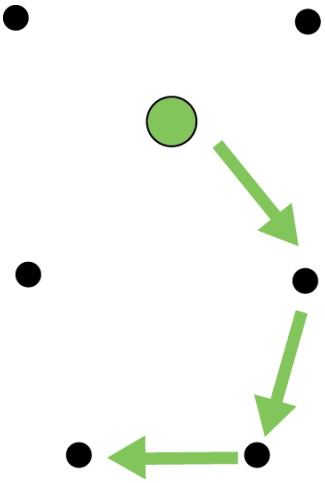
Zigbee source routing helps solve these problems. In contrast to many-to-one routing that establishes routing paths from many devices to one data collector, source routing allows the collector to store and specify routes for many remotes.

To use source routing, a device must use the API mode, and it must send periodic many-to-one route request broadcasts (AR command) to create a many-to-one route to it on all devices. When remote devices send RF data using a many-to-one route, they first send a route record transmission. The route record transmission is unicast along the many-to-one route until it reaches the data collector. As the route record traverses the many-to-one route, it appends the 16-bit address of each device in the

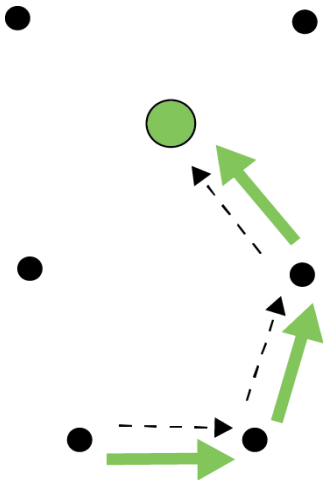
route into the RF payload. When the route record reaches the data collector, it contains the address of the sender, and the 16-bit address of each hop in the route. The data collector can store the routing information and retrieve it later to send a source routed packet to the remote as shown in the following images.



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



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



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

Acquire source routes

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

1. If the application on remote devices periodically sends data to the data collector, each transmission forces a route record to occur.
2. The data collector can issue a network discovery command (**ND** command) to force all XBee devices to send a network discovery response. A route record prefaces each network discovery response.
3. You can enable periodic I/O sampling on remotes to force them to send data at a regular rate. A route record prefaces each I/O sample. For more information, see Analog and digital I/O lines.
4. If the **NI** string of the remote device is known, the **DN** command can be issued with the **NI** string of the remote in the payload. The remote device with a matching **NI** string would send a route record and a DN response.

Store source routes

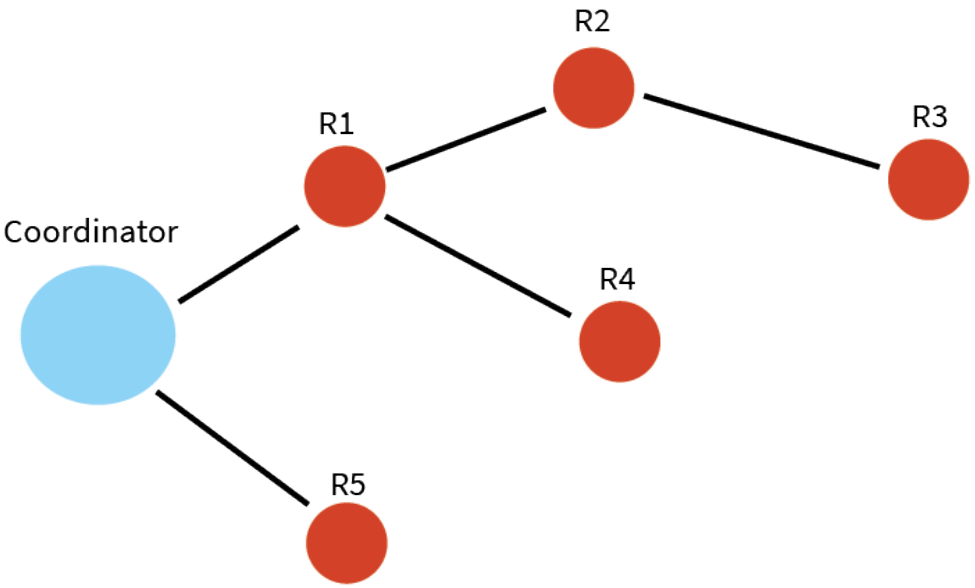
When a data collector receives a route record, it sends it out the serial port as a Route Record Indicator - 0xA1. To use source routing, the application receives these frames and stores the source route information.

Send a source routed transmission

To send a source routed transmission, the application must send a Create Source Route - 0x21 to the XBee 3 Zigbee RF Module to create a source route in its internal source route table. After sending the Create Source Route frame, the application can send data transmission or remote command request frames as needed to the same destination, or any destination in the source route. Once data must be sent to a new destination (a destination not included in the last source route), the application must first send a new Create Source Route - 0x21.

Note If a Create Source Route API frame does not precede the data frames, you may encounter data loss.

The XBee 3 Zigbee RF Module can buffer one source route that includes up to 11 hops (excluding source and destination). For example, suppose a network exists with a coordinator and 5 routers (R1, R2, R3, R4, R5) with known source routes as shown in the following image.



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

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

The Create Source Route API frame would be:

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

Field composition

0x0012	length
0x21	API ID (create source route)
0x00	frame ID (set to 0 always)
0x0013A200 404A1234	64-bit address of R3 (destination)

0xEEFF	16-bit address of R3 (destination)
0x00	Route options (set to 0)
0x02	Number of intermediate devices in the source route
0xCCDD	Address of furthest device (1-hop from target)
0xAABB	Address of next-closer device
0x5C	Checksum (0xFF - SUM (all bytes after length))

Repair source routes

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

As mentioned previously, source routing must be used with many-to-one routing. A device that uses source routing must also send a periodic many-to-one broadcast in order to keep routes fresh. If a source route breaks, remote devices send in new route record transmissions to the data collector to provide it with a new source route. This requires that remote devices periodically send data transmissions into the data collector. For more information, see Acquire source routes.

Retries and acknowledgments

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

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

Note Refer to the Zigbee specification for more details.

Disable MTO routing

To disable MTO (many-to-one) routing in a network, first reconfigure the **AR** setting on the aggregator and then broadcast a network wide power reset to rebuild the routing tables.

1. Set **AR** on the aggregator to **0xFF**.
2. Complete an **AC** command to enact the change.
3. Complete a **WR** command if the saved configuration setting value for **AR** is not 0xFF.

This ends the periodic broadcast of aggregator messages if the previous setting was 0x01 - 0xFE, and prevents a single broadcast after a power reset if the previous setting was 0x00. Broadcast a **FR** remote command to the network and wait for the network to reform. This removes the aggregator's status as an aggregator from the network's routing tables so that no more route records will be sent to the aggregator.

Disable route records

If an aggregator collects route records from the nodes of the network and no longer needs route records sent (which consume network throughput) :

1. Set Bit 6 of **DO** to Enable High RAM Concentrator mode. High RAM mode means the aggregator has sufficient memory to hold route records for its potential destinations.
2. Set **AR** to 0x00 for a one-time broadcast (which some nodes might miss), or a value in the range of 0x01 to 0xFE (in units of 10 seconds) to periodically send a broadcast to inform the network that the aggregator is operating in High RAM Concentrator mode and no longer needs to receive route records.
3. Use Create Source Route - 0x21 to load the route record for a destination into the local device's source route table.
4. Send a unicast to the destination. The route record embeds in the payload and determines the sequence of routers to use in transmitting the unicast to the destination. After receiving the unicast, the destination no longer sends route records to the aggregator, now that it has confirmed the High RAM Concentrator aggregator 'knows' its route record.

Clear the source route table

To clear the source route table, change the **AR** setting from a non-0xFF setting to 0xFF and complete an **AC** command. To re-establish periodic aggregator broadcasts, change the **AR** setting to a non-0xFF setting and complete an **AC** command.

Encrypted transmissions

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

Maximum RF payload size

The maximum payload size on the XBee 3 Zigbee RF Module is a function of the following:

- Message type: broadcast or unicast
- AES encryption (**EE** command)
- APS security (**TO** bit 4)
- Secure Session (**TO** bit 5)
- Source Routing

The maximum payload size of a single packet is:

Message type	Unicast	Broadcast
Unencrypted (**EE** = **0**)	84 bytes	92 bytes
Encrypted (**EE** = **1**)	66 bytes	74 bytes
APS Security (**EE** = **1**, **TO** bit 5)	57 bytes	N/A

When operating in Transparent mode (**AP** = **0**), all outgoing transmissions are sent as non-fragmented messages.

When sending a unicast transmission in API mode or through MicroPython, the maximum payload is 255 bytes. If the combination of payload and optional APS security overhead is too high, the message fragments into a maximum of five fragments. The firmware encrypts and transmits each fragment separately. The destination device reassembles the fragments into a full message.

Broadcast transmissions are sent as non-fragmented messages and cannot use APS security.

- Enabling encryption (**EE** = **1**) reduces maximum payload size by 18 bytes.
- Enabling APS security (**TO** bit 5) reduces maximum payload size by 9 bytes.
- Enabling Secure Session (**TO** bit 4) reduces maximum payload size by 5 bytes.

Using source routing will further reduce payload size depending on how many hops are being traversed. When an aggregator (**AR** < **0xFF**) sends a source-routed message, it embeds the route into the message as overhead, or into each fragment of the message if fragmentation is necessary. If you use APS security (**EE 1**, Tx Option 0x20), it reduces the number further.

The route overhead is two bytes plus two bytes per hop. The bytes are:

- One byte for the number of hops.
- One byte is an index into the route list that increments in value at each hop.
- For each hop, two bytes are used for the 16-bit network address of each routing device.

Aggregator source-routed payload maximums do not apply to messages that are sourced by non-aggregator nodes (**AR** = **0xFF**).

The following table shows the aggregator source-routed payload maximums (in bytes) as a function of hops and APS security:

Hops	Maximum payload	Maximum APS-encrypted payload
1	255	255
2	255	255
3	255	255
4	255	255
5	255	255
6	255	215
7	250	205
8	240	195
9	230	185
10	220	175
11	210	165
12	200	155

13	190	145
14	180	135
15	170	125
16	160	115
17	150	105
18	140	95
19	130	85
20	120	75
21	110	65
22	100	55
23	90	45
24	80	35
25	70	25

Throughput

Throughput in a Zigbee network can differ by a number of variables, including:

- Number of hops
- Encryption enabled/disabled
- Sleeping end devices
- Failures/route discoveries

ZDO transmissions

Zigbee defines a Zigbee device objects layer (ZDO) that provides device and service discovery and network management capabilities.

Cluster name	Cluster ID	Description
Network Address Request	0x0000	Request a 16-bit address of the radio with a matching 64-bit address (required parameter).
Active Endpoints Request	0x0005	Request a list of endpoints from a remote device.
LQI Request	0x0031	Request data from a neighbor table of a remote device.
Routing Table Request	0x0032	Request to retrieve routing table entries from a remote device.

Cluster name	Cluster ID	Description
Network Address Response	0x8000	Response that includes the 16-bit address of a device.
LQI Response	0x8031	Response that includes neighbor table data from a remote device.
Routing Table Response	0x8032	Response that includes routing table entry data from a remote device.

Refer to the Zigbee specification for a detailed description of all Zigbee device profile services.

Send a ZDO command

When operating in API mode, ZDO commands can be sent as the payload of an explicit transmit API frame (0x11). The outgoing ZDO command must be formatted properly with the correct byte order and endianness observed. In order to receive responses to outgoing ZDO commands, you need to enable ZDO pass-through using AO (API Options).

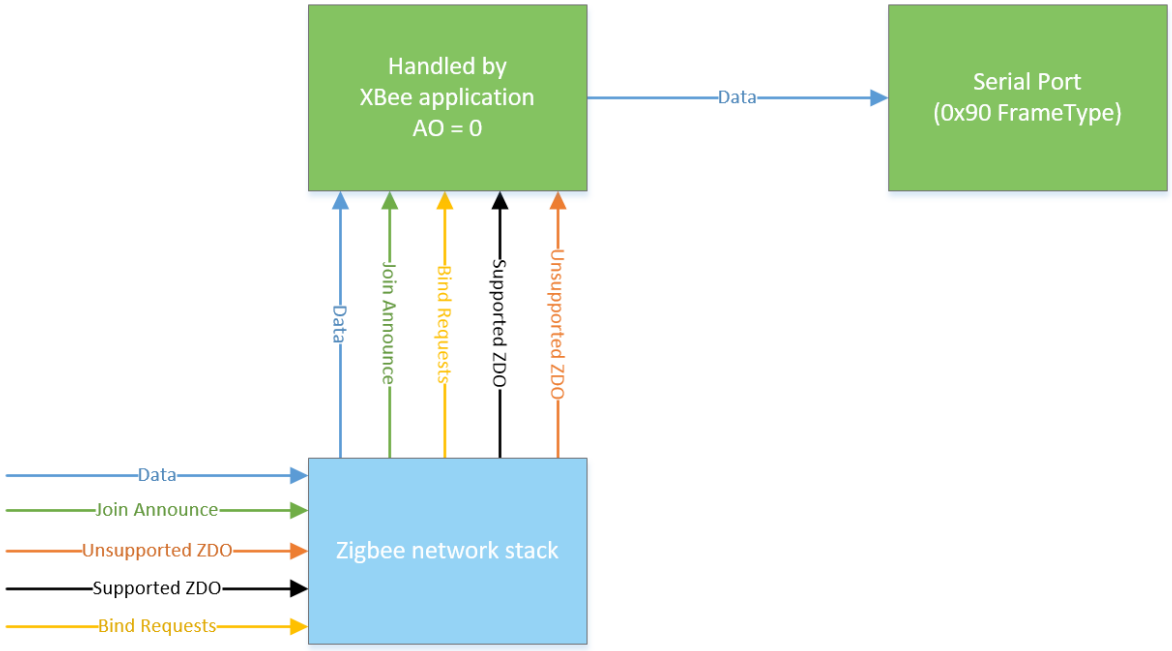
To send a ZDO command:

1. Set the source and destination endpoints and profile ID to **0**.
2. Set the cluster ID to match the cluster ID of the appropriate service. For example, to send an active endpoints request, set the cluster ID to **0x0005**.
3. The first byte of payload in the API frame is an application sequence number (transaction sequence number) that can be set to any single byte value. The first byte of the ZDO response uses this same value.
4. All remaining payload bytes must be set as required by the ZDO. All multi-byte values must be sent in little endian byte order.

Receiving ZDO command and responses

Incoming ZDO commands and responses are handled by the XBee application by default. In order to receive and work with incoming ZDO commands, you must configure the device to pass ZDOs to the serial port instead of the XBee handling them. AO (API Options) is used to control this.

When operating in API mode and with **AO** set to **0**, the output format for received data packets is Digi's native 0x90 receive frame format. In this configuration, the XBee application will handle and respond to any incoming ZDO requests. For unsupported ZDO commands, the XBee 3 Zigbee RF Module will respond with: **ZDO not supported**. The following figure shows **AO** set to **0**.

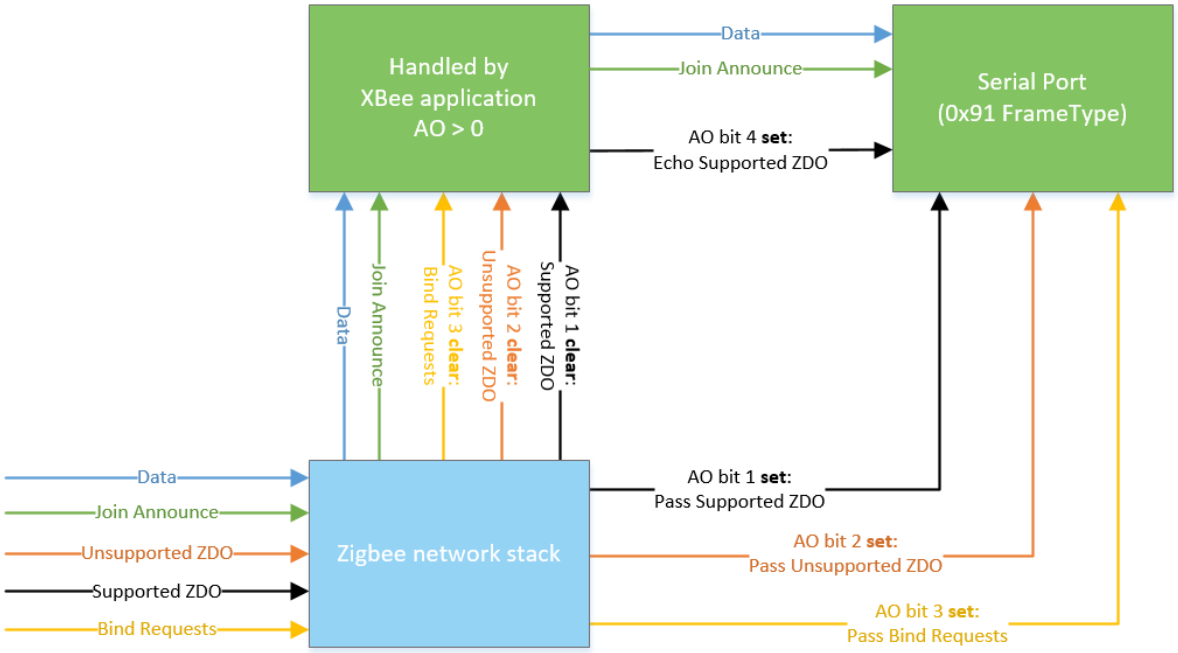


When **AO** is non-zero, the API frame format for received data packets is an explicit 0x91 receive frame. This frame contains the additional fields necessary to interpret ZDO messages.

Bits 1, 2, and 3 of the **AO** command dictate the routing of incoming ZDO messages. When these bits are cleared, the XBee 3 Zigbee RF Module will handle and respond to ZDO commands. When these bits are set, then Supported ZDO, Unsupported ZDO, and/or Bind Requests are passed through the UART and the XBee device will not respond.

Bit 4 of the **AO** command will allow any supported ZDO commands that the XBee application handles to be echoed out of the serial port. This is useful as a diagnostic tool to identify when the XBee 3 Zigbee RF Module is responding to ZDO commands and what types.

Setting bit 5 of **AO** will suppress all ZDO output and disable pass through. Setting bit 5 will behave as if bits 1, 2, and 3 are 0 (XBee device handles incoming requests). This is useful if you want to use the 0x91 receive frame, but only emit Digi-specific messages out of the serial port. The following figure shows **AO** set to a non-zero value.



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

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

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

API Frame

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

Field composition

0x0016	length
0x11	Explicit transmit request
0x01	Frame ID (set to a non-zero value to enable the transmit status message, or set to 0 to disable)
0x0013A200 40401234	64-bit address of the remote
0xFFFE	16-bit address of the remote (0xFFFE = unknown). Optionally, set to the 16-bit address of the destination if known.

0x00	Source endpoint
0x00	Destination endpoint
0x0031	Cluster ID (LQI Request, or Neighbor table request)
0x0000	Profile ID (Zigbee device profile)
0x00	Broadcast radius
0x00	Tx Options
0x76	Transaction sequence number
0x00	Required payload for LQI request command
0xCE	Checksum (0xFF - SUM (all bytes after length))

Description

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

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

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

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

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

API frame

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

Field composition

0x001F	length
0x11	Explicit transmit request
0x01	Frame ID (set to a non-zero value to enable the transmit status message, or set to 0 to disable)
0x00000000 0000FFFF	64-bit address for a broadcast transmission
0xFFFE	Set to this value for a broadcast transmission
0x00	Source endpoint
0x00	Destination endpoint
0x0000	Cluster ID (Network Address Request)

0x0000	Profile ID (Zigbee device profile)
0x00	Broadcast radius
0x00	Tx Options
0x44	Transaction sequence number
0x34124040 00A21300 00 00	Required payload for Network Address Request command
0x33	Checksum (0xFF - SUM (all bytes after length))

Description

This API frame sends a broadcast ZDO Network Address Request to obtain the 16-bit address of a device with a 64-bit address of 0x0013A200 40401234. We inserted the bytes for the 64-bit address in little endian byte order. You must insert data for all multi-byte fields in the API payload of a ZDO command in little endian byte order. You must set the **AO** command correctly on an API device to enable the explicit API receive frames to receive the ZDO response.

Support ZDOs with the XBee API

The Zigbee Device Profile is a management and discovery service layer supported on all Zigbee devices. Like all other profiles, the Zigbee Device Profile defines a set of clusters that can be used to perform a variety of advanced network management and device discovery operations. Since the Zigbee Device Profile is supported to some extent on all Zigbee devices, many Zigbee Device Profile cluster operations can be performed on a variety of Zigbee devices, regardless of the stack or chipset manufacturer.

The Zigbee Device Profile has an application profile identifier of 0x0000. All Zigbee devices support a reserved endpoint called the Zigbee Device Objects (ZDO) endpoint. The ZDO endpoint runs on endpoint 0 and supports clusters in the Zigbee Device Profile. All devices that support the Zigbee Device Profile clusters support endpoint 0.

ZDO services include the following features:

- View the neighbor table on any device in the network
- View the routing table on any device in the network
- View the end device children of any device in the network
- Obtain a list of supported endpoints on any device in the network
- Force a device to leave the network
- Enable or disable the permit-joining attribute on one or more devices

Support the ZDP with the XBee API

The XBee API provides a simple interface to the Zigbee Device Objects endpoint. The explicit transmit API frame (API ID 0x11) allows data transmissions to set the source and destination endpoints, cluster ID, and profile ID. ZDO commands can be sent by setting the source and destination endpoints to the ZDO endpoint (0x00), the profile ID to the Zigbee Device Profile ID (0x0000), and the cluster ID to the appropriate ZDO cluster ID.

The data payload must contain a sequence number as the first byte (transaction sequence number), followed by all required payload bytes for the ZDO. Multi-byte fields must be sent in little endian byte order.

To receive ZDO commands and responses, the AO (API Output) command must be set to 1. This

enables the explicit receive API frame (API ID 0x91) which indicates the source and destination endpoints, cluster ID, and profile ID.

ZDO clusters

The following section outlines common ZDO commands including the following:

Note Not all of these commands are supported by the XBee. Unsupported commands can be handled by a host processor or in MicroPython by enabling unsupported ZDO passthrough—See AO (API Options).

ZDO command	Cluster ID	Supported by the XBee
Network (16-bit) Address Request	0x0000	Yes
Network (16-bit) Address Response	0x8000	Yes
IEEE (64-bit) Address Request	0x0001	Yes
IEEE (64-bit) Address Response	0x8001	Yes
Node Descriptor Request	0x0002	Yes
Node Descriptor Response	0x8002	Yes
Simple Descriptor Request	0x0004	No
Simple Descriptor Response	0x8004	No
Active Endpoints Request	0x0005	Yes
Active Endpoints Response	0x8005	Yes
Match Descriptor Request	0x0006	No
Match Descriptor Response	0x8006	No
End Device Bind Request	0x0020	Yes
End Device Bind Response	0x8020	Yes
Unbind Request	0x0022	Yes
Unbind Response	0x8022	Yes
Management LQI (Neighbor Table) Request	0x0031	Yes
Management LQI (Neighbor Table) Response	0x8031	Yes
Management Rtg (Routing Table) Request	0x0032	Yes
Management Rtg (Routing Table) Response	0x8032	Yes
Management Leave Request	0x0034	Yes
Management Leave Response	0x8034	Yes
Management Permit Join Request	0x0036	Yes
Management Permit Join Response	0x8036	Yes

ZDO command	Cluster ID	Supported by the XBee
Management Network Update Request	0x0038	No
Management Network Update Response	0x8038	No

Network Address Request

Cluster ID

0x0000

Description

Broadcast transmission used to discover the 16-bit (network) address of a remote device with a matching 64-bit address.

Field Name	Size (bytes)	Description
Sequence number	1	Transaction sequence number—arbitrarily chosen.
IEEE Address	8	64-bit address of a device in the network whose 16-bit (network) address is being discovered.
Request Type	1	0x00 – Single device response. Only the device with a matching IEEE address responds. 0x01 – Extended response. The device with a matching IEEE address responds AND sends a list of the 16-bit addresses of devices in its associated device list starting at 'Start Index' until the next entry will not fit in the data payload.
Start Index	1	Indicates the starting index in the associated device list to return 16-bit addresses. Only used if extended response is requested.

Network Address Response

Cluster ID

0x8000

Description

Indicates the 16-bit (network) address of a remote whose 64-bit address matched the address in the request. If an extended response was requested, this will also include the 16-bit addresses of devices in the associated device list.

Field Name	Size (bytes)	Description
Sequence number	1	Transaction sequence number used in the request.
Status	1	

Field Name	Size (bytes)	Description
IEEE Address	8	Indicates the 64-bit address of the responding device.
Network Address	2	Indicates the 16-bit address of the responding device.
Number of Addresses	0/1	Returns the number of addresses in the packet. Byte not included in response if an extended response was not requested.
Start Index	0/1	Starting index into the associated device list for this report. Multiple requests might be necessary to read all devices in the list.
Network Addresses of Associated Device List	Variable	List of all 16-bit addresses in the associated device list.

IEEE Address Request

Cluster ID

0x0001

Description

Unicast transmission used to discover the 64-bit (IEEE) address of a remote device with a matching 16-bit address.

Field Name	Size (bytes)	Description
Sequence number	1	Transaction sequence number (arbitrarily chosen).
Network Address	2	16-bit address of a device in the network whose 64-bit (network) address is being discovered.
Request Type	1	0x00 – Single device response. (Only the device with a matching IEEE address responds.) 0x01 – Extended response. (The device with a matching IEEE address responds AND sends a list of the 16-bit addresses of devices in its associated device list starting at 'Start Index' until the next entry won't fit in the data payload.
Start Index	1	Indicates the starting index in the associated device list to return 16-bit addresses. Only used if extended response is requested.

IEEE Address Response

Cluster ID

0x8001

Description

Indicates the 64-bit (IEEE) address of a remote whose 16-bit address matched the address in the request. If an extended response was requested, this will also include the 16-bit addresses of devices in the associated device list.

Field Name	Size (bytes)	Description
Sequence number	1	Transaction sequence number used in the request.
Status	1	
IEEE Address	8	Indicates the 64-bit address of the responding device.
Network Address	2	Indicates the 16-bit address of the responding device.
Number of Addresses	0/1	Returns the number of addresses in the packet. Byte not included in response if an extended response was not requested.
Start Index	0/1	Starting index into the associated device list for this report. Multiple requests might be necessary to read all devices in the list.
Network Addresses of Associated Device List	Variable	List of all 16-bit addresses in the associated device list.

Node Descriptor Request

Cluster ID

0x0002

Description

Transmission used to discover the node descriptor of a device with a matching 16-bit address.

Field Name	Size (bytes)	Description
Sequence number	1	Transaction sequence number (arbitrarily chosen).
Network Address	2	16-bit address of a device in the network whose node descriptor is being requested.

Node Descriptor Response

Cluster ID

0x8002

Description

Indicates the node descriptor of the device.

Field Name	Size (bytes)	Description
Sequence number	1	Transaction sequence number used in the request.
Status	1	
Network Address	2	Indicates the 16-bit address of the responding device.
Node Descriptor	Variable	See node descriptor below.

Node Descriptor

Name	Size (bits)	Description
Logical Type	3	Indicates the logical device type: 000 – Coordinator 001 – Router 010 – End device
Complex Descriptor Available	1	1. Complex descriptor not supported 2. Complex descriptor supported
User Descriptor Available	1	1. User descriptor not supported 2. User descriptor supported
Reserved	3	
APS flags	3	Not supported. Set to 0.
Frequency Band	5	bit0 – 868 MHz bit1 – Reserved bit2 – 900 MHz bit3 – 2.4 GHz bit4 – Reserved
MAC capability flags	8	Bit0 – Alternate PAN coordinator Bit1 – Device Type Bit2 – Power source Bit3 – Receiver on when idle Bit4-5 – Reserved Bit6 – Security capability Bit7 – Allocate address
Manufacturer Code	16	Indicates the manufacturer's code assigned by the Zigbee Alliance.
Maximum Buffer Size	8	Maximum size in bytes, of a data transmission (including APS bytes).
Maximum incoming transfer size	16	Maximum number of bytes that can be received by the node.

Name	Size (bits)	Description
Server mask	16	
Maximum outgoingtransfer size	16	Maximum number of bytes that can be transmitted by this device, including fragmentation.
Descriptor capability field	8	Bit0 – Extended active endpoint list available Bit1 – Extended simple descriptor list available

Simple Descriptor Response

Cluster ID

0x0004

Description

Transmission used to discover the simple descriptor of a device with a matching 16-bit address.

Field Name	Size (bytes)	Description
Sequence number	1	Transaction sequence number (arbitrarily chosen).
Network Address	2	16-bit address of a device in the network whose simple descriptor is being requested.
Endpoint	1	The endpoint on the destination from which to obtain the simple descriptor.

Simple Descriptor Response

Cluster ID

0x8004

Description

Indicates the simple descriptor of the device.

Field Name	Size (bytes)	Description
Sequence number	1	Transaction sequence number used in the request.
Status	1	
Network Address	2	Indicates the 16-bit address of the responding device.
Length	1	Length of the simple descriptor.
Simple Descriptor	Variable	See simple descriptor below.

Simple Descriptor

Name	Size (bits)	Description
Endpoint	8	The endpoint on the node to which this descriptor refers.
Application profile ID	16	The profile ID supported on this endpoint.
Application device ID	16	Specifies the device description identifier supported on the device.
Application device version	4	The version of the device description supported on this endpoint.
Reserved	4	
Input cluster count	8	The number of input clusters supported on this endpoint.
Input cluster list	Variable	The list of input clusters supported on this endpoint. Each cluster is 2 bytes in size. This field is not included if the input cluster count is 0.
Output cluster count	8	The number of output clusters supported on this endpoint.
Output cluster list	Variable	The list of output clusters supported on this endpoint. Each cluster is 2 bytes in size. This field is not included if the output cluster count is 0.

Active Endpoints Request

Cluster ID

0x0005

Description

Transmission used to discover the active endpoints on a device with a matching 16-bit address.

Field Name	Size (bytes)	Description
Sequence number	1	Transaction sequence number (arbitrarily chosen)
Network Address	2	16-bit address of a device in the network whose active endpoint list being requested.

Active Endpoints Response

Cluster ID

0x8005

Description

Indicates the list of active endpoints supported on the device.

Field Name	Size (bytes)	Description
Sequence number	1	Transaction sequence number used in the request.
Status	1	
Network Address	2	Indicates the 16-bit address of the responding device.
Active Endpoint Count	1	Number of endpoints in the following endpoint list.
Active Endpoint List	Variable	List of endpoints supported on the destination device. One byte per endpoint.

Match Descriptor Request

Cluster ID

0x0006

Description

Broadcast or unicast transmission used to discover the device(s) that supports a specified profile ID and/or clusters.

Field Name	Size (bytes)	Description
Sequence number	1	Transaction sequence number (arbitrarily chosen).
Network Address	2	16-bit address of a device in the network whose power descriptor is being requested.
Profile ID	2	Profile ID to be matched at the destination.
Number of Input Clusters	1	The number of input clusters in the In Cluster List for matching. Set to 0 if no clusters supplied.
Input Cluster List	2 * Number of Input Clusters	List of input cluster IDs to be used for matching.
Number of Output Clusters	1	The number of output clusters in the Output Cluster List for matching. Set to 0 if no clusters supplied.

Field Name	Size (bytes)	Description
Output Cluster List	2 * Number of Input Clusters	List of output cluster IDs to be used for matching.

Match Descriptor Response

Cluster ID

0x8006

Description

If a descriptor match is found on the device, this response contains a list of endpoints that support the request criteria.

Field Name	Size (bytes)	Description
Sequence number	1	Transaction sequence number used in the request.
Status	1	
Network Address	2	Indicates the 16-bit address of the responding device.
Length	1	The number of endpoints on the remote device that match the request criteria.
Match List	Variable	List of endpoints on the remote that match the request criteria.

End Device Bind Request

Cluster ID

0x0020

Description

Unicast transmission to the coordinator for binding devices.

Field Name	Size (bytes)	Description
Sequence number	1	Transaction sequence number (arbitrarily chosen).
Network Address	2	Node ID of source device.
Extended Address	8	64 bit Address of the source device.
Endpoint	1	Targeted binding endpoint.
Profile ID	2	Profile of the targeted binding device.
Number of input Clusters	1	The number of targeted input cluster(s).

Field Name	Size (bytes)	Description
Input Cluster ID	2 bytes for each entry	The targeted input cluster(s).
Number of output Clusters	1	The number of targeted output cluster(s).
Output Cluster ID	2 bytes for each entry	The targeted output cluster(s).

End Device Bind Response

Cluster ID

0x8020

Description

Response to end device request.

Field Name	Size (bytes)	Description
Sequence number	1	Transaction sequence number used in the request.
Status	1	Status of End device request.

Bind/Unbind Request

Cluster ID

0x0021/0x0022

Description

Bind and Unbind Requests have the same format

Field Name	Size (bytes)	Description
Sequence number	1	Transaction sequence number (arbitrarily chosen)
Extended Address	8	64 bit Source Address of the device sending the request
Source Endpoint	1	The endpoint needed for binding
Cluster ID	2	The cluster ID needed for binding. If doing unbind will be this cluster ID if previously used the Bind request, otherwise it will be the Output Cluster ID used in End Device Bind request.

Field Name	Size (bytes)	Description
Address Mode	1	A fixed value indicating using one of the following address modes: 0x01 - Destination group (will require 2 additional bytes, see Note) 0x03 - Destination extended address and endpoint (9 additional bytes, see Note) **Note** Tables below has the additional fields for the payload.

Address mode 0x01: Unbind Destination Group

Field Name	Size (bytes)	Description
Destination group	2	Destination group used in the Bind Request Cluster ID 0x0021.

Address mode 0x03: Unbind Destination extended address and endpoint

Field Name	Size (bytes)	Description
Extended Address	8	64 bit Address of the destination device to be bind/unbind together. If doing unbind will be address used in the Bind or End Device Bind request.
Endpoint	1	The endpoint needed for binding. If doing unbind will be endpoint used in the Bind or End Device Bind request.

Bind/Unbind Response

Cluster ID

0x8021/0x8022

Description

Response to either the Bind or Unbind request.

Field Name	Size (bytes)	Description
Sequence number	1	Transaction sequence number used in the request.
Status	1	Status from ether the Bind or Unbind request.

Management LQI (Neighbor Table) Request

Cluster ID

0x0031

Description

Unicast transmission used to cause a remote device to return the contents of its neighbor table.

Field Name	Size (bytes)	Description
Sequence number	1	Transaction sequence number (arbitrarily chosen)
Start Index	1	Start index in the neighbor table to return neighbor entries. The response cannot include more than 2-3 entries. Multiple LQI requests may be required to read the entire neighbor table.

Management LQI (Neighbor Table) Response

Cluster ID

0x8031

Description

Indicates the neighbor table contents of the device.

Field Name	Size (bytes)	Description
Sequence number	1	Transaction sequence number used in the request.
Status	1	
Neighbor Table Entries	1	The total number of neighbor table entries.
Start Index	1	The starting point in the neighbor table.
Network Table List Count	1	The number of neighbor table entries in this response.
Neighbor Table List	Variable	A list of neighbor table entries.

Neighbor Table Entry

Name	Size (bits)	Description
Extended PAN ID	64	The 64-bit extended PAN ID of the neighboring device.
ExtendedAddress	64	64-bit address of the neighboring device.
NetworkAddress	16	The 16-bit address of the neighboring device.
Device Type	2	The type of neighbor: 0x0 – Zigbee coordinator 0x1 – Zigbee router 0x2 – Zigbee end device 0x3 – Unknown
Receiver On When Idle	2	Indicates if the neighbor's receiver is enabled during idle times. 0x0 – Receiver is off 0x1 – Receiver is on 0x02 – Unknown

Name	Size (bits)	Description
Relationship	3	The relationship of the neighbor with the remote device: 0x0 – Neighbor is the parent 0x1 – Neighbor is a child 0x2 – Neighbor is a sibling 0x3 – None of the above 0x4 – Previous child
Reserved	1	Set to 0.
Permit Joining	2	Indicates if the neighbor is accepting join requests. 0x0 – Neighbor not accepting joins 0x1 – Neighbor is accepting joins 0x2 – Unknown
Reserved	6	Set to 0.
Depth	8	The tree depth of the neighbor device. A value of 0x00 indicates the device is the Zigbee coordinator for the network.
LQI	8	The estimated link quality of data transmissions from this neighboring device.

Management Rtg (Routing Table) Request

Cluster ID

0x0032

Description

Unicast transmission used to cause a remote device to return the contents of its routing table.

Field Name	Size (bytes)	Description
Sequence number	1	Transaction sequence number (arbitrarily chosen)
Start Index	1	Start index in the routing table to return routing table entries. The response cannot include more than a handful of entries. Multiple routing table requests may be required to read the entire routing table.

Management Rtg (Routing Table) Response

Cluster ID

0x8032

Description

Indicates the routing table contents of the device.

Field Name	Size (bytes)	Description
Sequence number	1	Transaction sequence number used in the request.
Status	1	
Routing Table Entries	1	The total number of routing table entries.
Start Index	1	The starting point in the routing table.
Routing Table List Count	1	The number of routing table entries in this response.
Routing Table List	Variable	A list of routing table entries.

Routing Table Entry

Name	Size (bits)	Description
Destination Address	16	The 16-bit address of the destination device.
Status	3	Status of the route: 0x0 – Active 0x1 – Discovery Underway 0x2 – Discovery Failed 0x3 – Inactive 0x4 – Validation Underway
Memory Constrained Flag	1	Indicates if the device is a low-memory concentrator.
Many-to-One Flag	1	Flag indicating the destination is a concentrator (ssued a many-to-onerequest).
Route Record Required	1	Flag indicating if a route record message should be sent to the device prior to the next data transmission.
Reserved	2	
Next-hop Address	16	16-bit address of the next hop.

Management Leave Request

Cluster ID

0x0034

Description

Transmission used to cause a remote device to leave the network.

Field Name	Size (bytes)	Description
Sequence number	1	Transaction sequence number (arbitrarily chosen)
Device Address	8	Address of the device the command is addressed to. See section 3.2.2.1.6 for details.
Options	1	Bitfield: 0x01 – Rejoin—If set, the device is asked to rejoin the network. 0x02 – Remove Children—If set, the device should remove its children.

Management Leave Response

Cluster ID

0x8034

Description

Indicates the status of a leave request.

Field Name	Size (bytes)	Description
Sequence number	1	Transaction sequence number used in the request
Status	1	Indicates the status of a leave request.

Management Permit Join Request

Cluster ID

0x0036

Description

Unicast or broadcast transmission used to cause a remote device or devices to enable joining for a time.

Field Name	Size (bytes)	Description
Sequence number	1	Transaction sequence number (arbitrarily chosen)
Permit Duration	1	Specifies the time that joining should be enabled (in seconds). If set to 0xFF, joining is enabled permanently.
Trust Center Significance	1	If set to 1 and the remote is a trust center, the command affects the trust center authentication policy. Otherwise, it has no effect.

Management Permit Joining Response

Cluster ID

0x8036

Description

Indicates the status of a permit joining request.

Field Name	Size (bytes)	Description
Sequence number	1	Transaction sequence number used in the request
Status	1	Indicates the status of a permit joining request.

Management Network Update Request

Cluster ID

0x0038

Description

Unicast transmission used to cause a remote device to do one of several things:

- Update the channel mask and network manager address (if scan duration = 0xFF)
- Change the network operating channel (if scan duration = 0xFE)
- Request to scan channels and report the results (if scan duration < 6)

Field Name	Size (bytes)	Description
Sequence number	1	Transaction sequence number (arbitrarily chosen)
Scan Channels	4	Bitmap indicating the channel mask that should be scanned. Examples (big endian byte order): Channel 0x0B = 0x800 Channel 0x10 = 0x10000 Channel 0x1A = 0x4000000 All Channels (0x0B – 0x1A) = 0x07FFF800
Scan Duration	1	Set as described above to invoke the desired command.
Scan Count	0/1	If scan duration < 6, specifies the number of energy scans to conduct and report. This can result in multiple responses being sent.
Network Update ID	0/1	Set by the network channel manager
Network Manager Address	0/2	If scan duration = 0xFF, indicates the network address of the network manager—who has network manager bit set in its node descriptor.

Management Network Update Response

Cluster ID

0x8038

Description

Indicates the RF conditions near the device.

Field Name	Size (bytes)	Description
Sequence number	1	Transaction sequence number used in the request.
Status	1	Status of the Management Network Update notify command.
Scanned Channels	4	List of channels scanned by the request.
Total Transmissions	2	Count of the total transmissions reported by the device.
Transmission Failures	2	Sum of the transmission failures reported by the device.
ScannedChannels List Count	1	The number of records contained in the energy values list.
Energy Values	Variable	The result of an energy measurement made on the scanned channels, one byte per energy measurement. 0xFF – Too much interference on the channel.

API example 1

Send a broadcast transmission to discover the 16-bit address of a device with a 64-bit address of 0x0013A200 44332211 using the Network Address Request ZDO—cluster ID = 0x0000. Format the command to also discover the 16-bit addresses of its children—if any.

1. To send this command, use the following fields:

0x11	API ID—transmit request.
0x00	Frame ID—set to 0 to disable transmit status.
0x00000000	0000FFFF 64-bit address for a broadcast transmission. 0xFFFE 16-bit address for a broadcast transmission.
0x00	Source endpoint—ZDO endpoint.
0x00	Destination endpoint—ZDO endpoint.
0x0000	Cluster ID—Network Address Request.
0x0000	Profile ID—Zigbee Device Profile ID.
0x00	Broadcast radius.
0x00	Transmit options.

2. In the ZDO payload, set a transaction sequence number.
3. Follow with the required payload for the network address request ZDO. The following bytes will be inserted into the data payload portion of the API frame:

0x01	Transaction sequence number—arbitrarily chosen.
0x44332211 00A21300	IEEE (64-bit) address of target device—little-endian byte order.
0x01	Request type—extended device request.
0x00	Start Index.

4. Calculate the length and checksum bytes to construct the final API frame.

Length	Count all bytes after the length bytes, excluding the checksum.
Checksum	(0xFF – SUM—all bytes after length).

Final API frame

7E 00 1F 11 00 00000000 0000FFFF FFFE 00 00 0000 0000 00 00 01 44332211 00A21300 01 00 92

API example 2

Send a broadcast transmission to discover the 64-bit address of a device with a 16-bit address of 0x3344 using the IEEE Address Request ZDO—cluster ID = 0x0001.

1. To send this command, use the following fields:

0x11	API ID—transmit request.
0x00	Frame ID—set to 0 to disable transmit status.
0x00000000	0000FFFF 64-bit address for a broadcast transmission. 0xFFFE 16-bit address for a broadcast transmission.
0x00	Source endpoint—ZDO endpoint.
0x00	Destination endpoint—ZDO endpoint.
0x0001	Cluster ID—IEEE Address Request.
0x0000	Profile ID—Zigbee Device Profile ID.
0x00	Broadcast radius.
0x00	Transmit options.

2. In the ZDO payload, set a transaction sequence number.
3. Follow with the required payload for the network address request ZDO. The following bytes will be inserted into the data payload portion of the API frame:

0x02	Transaction sequence number—arbitrarily chosen.
0x4433	Network (16-bit) address of target device—little-endian byte order.
0x01	Request type—single device request.
0x00	Start Index.

4. Calculate the length and checksum bytes to construct the final API frame.

Length	Count all bytes after the length bytes, excluding the checksum.
Checksum	(0xFF – SUM—all bytes after length).

Final API frame

7E 00 19 11 00 00000000 0000FFFF FFFE 00 00 0001 0000 00 00 02 4433 00 00 79

API example 3

Send a broadcast transmission to discover the node descriptor of a device with a 16-bit address of 0x3344.

1. To send this command, use the following fields:

0x11	API ID—transmit request.
0x00	Frame ID—set to 0 to disable transmit status.
0x00000000	0000FFFF 64-bit address for a broadcast transmission. 0xFFFE 16-bit address for a broadcast transmission.
0x00	Source endpoint—ZDO endpoint.
0x00	Destination endpoint—ZDO endpoint.
0x0002	Cluster ID—Node Descriptor Request.
0x0000	Profile ID—Zigbee Device Profile ID.
0x00	Broadcast radius.
0x00	Transmit options.

2. In the ZDO payload, set a transaction sequence number.
3. Follow with the required payload for the network address request ZDO. The following bytes will be inserted into the data payload portion of the API frame:

0x03	Transaction sequence number—arbitrarily chosen.
0x4433	Network (16-bit) address of target device—little-endian byte order.
0x01	Request type—single device request.
0x00	Start Index.

4. Calculate the length and checksum bytes to construct the final API frame.

Length	Count all bytes after the length bytes, excluding the checksum.
Checksum	(0xFF – SUM—all bytes after length).

Final API frame

7E 00 17 11 00 00000000 0000FFFF FFFE 00 00 0002 0000 00 00 03 4433 77

API example 4

Send a unicast data transmission to read the neighbor table of a router with 64-bit address 0x0013A200 40401234 using the LQI Request ZDO—cluster ID = 0x0031.

1. To send this command, use the following fields:

0x11	API ID—transmit request.
0x00	Frame ID—set to 0 to disable transmit status.
0x0013A200	40401234 64-bit address for a broadcast transmission. 0xFFFE 16-bit address for a broadcast transmission—0xFFFE if unknown.
0x00	Source endpoint—ZDO endpoint.
0x00	Destination endpoint—ZDO endpoint.
0x0031	Cluster ID—LQI Request.
0x0000	Profile ID—Zigbee Device Profile ID.
0x00	Broadcast radius.
0x00	Transmit options.

2. In the ZDO payload, set a transaction sequence number.
3. Follow with the required payload for the network address request ZDO. The following bytes will be inserted into the data payload portion of the API frame:

0x76	Transaction sequence number—arbitrarily chosen.
0x00	Start Index.

4. Calculate the length and checksum bytes to construct the final API frame.

Length	Count all bytes after the length bytes, excluding the checksum.
Checksum	(0xFF – SUM—all bytes after length).

Final API frame

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

API example 5

Send a unicast data transmission to have a remote router perform an energy scan on all channels using a ZDO Management Network Update Request—cluster ID = 0x0038. In this example, the 64-bit address of the router is 0x0013A200 40522BAA.

1. To send this command, use the following fields:

0x11	API ID—transmit request.
0x00	Frame ID—set to 0 to disable transmit status.
0x0013A200	40522BAA 64-bit address for a broadcast transmission. 0xFFFE 16-bit address for a broadcast transmission—0xFFFE if unknown.
0x00	Source endpoint—ZDO endpoint.
0x00	Destination endpoint—ZDO endpoint.
0x0038	Cluster ID—Management Network Update Request.
0x0000	Profile ID—Zigbee Device Profile ID.
0x00	Broadcast radius.
0x00	Transmit options.

2. In the ZDO payload, set a transaction sequence number.
3. Follow with the required payload for the network address request ZDO. The following bytes will be inserted into the data payload portion of the API frame:

0x01	Transaction sequence number—arbitrarily chosen.
0x00F8FF07	Scan channels—all 16 channels, little-endian byte order.
0x03	Scan duration.
0x02	Scan count—perform two energy scans.

4. The **Network Update ID** and **Network Manager Address** fields are not required for this operation.
5. Calculate the length and checksum bytes to construct the final API frame.

Length	Count all bytes after the length bytes, excluding the checksum.
Checksum	(0xFF – SUM—all bytes after length).

Final API frame

7E 001B 11 00 0013A200 40522BAA FFFE 00 00 0038 0000 00 00 01 00F8FF07 03 02 99

API example 6

Parse a Management Network Update Response received in response to API example 5 to extract energy data on the scan channels mask.

Recall that AO (API Options) must be set on an API device to receive ZDO responses. Suppose the following API frame is received.

API frame

7E 002D 91 0013A200 40522BAA 06FC 00 00 8038 0000 01 01 00 00F8FF07 1D00 0000 10 54 5E 69 5B 4B 48 44 48 55 55 57 46 51 41 44 4B 6E

Decoded API frame

0x7E	Start delimiter.
0x002D	Length.
0x91	Explicit receive API frame.
0x0013A200	40522BAA 64-bit address of the remote—who performed the energy scan. 0x06FC 16-bit address of the remote.
0x00	Source endpoint—ZDO endpoint.
0x00	Destination endpoint—ZDO endpoint.
0x8038	Cluster ID—Management network update notify.
0x0000	Profile ID—Zigbee Device Profile ID.
0x01	Rx options—packet was acknowledged.
0x010000F8FF071D00000010545E...41444B	Data payload.
0x6E	Checksum.

The data payload bytes can be interpreted as a ZDO management network update notify packet. Recall that the first byte in the data payload is a transaction sequence number that matches the sequence number of the request.

Data payload bytes (Management Network Update Response)

0x01	Transaction sequence number used in request.
0x00	Status (SUCCESS).
0x00F8FF07	Channel mask (16 channels enabled, represented in little endian byte order).
0x1D00	Total transmissions (0x001D = 29).
0x0000	Transmission failures.

0x10	Scanned channel count.
0x54	1st channel in channel mask energy level (channel 0x0B).
0x5E	2nd channel in channel mask energy level (channel 0x0C).
…	
0x4B	last channel in channel mask energy level (channel 0x1A).

In the Ember stack, to convert energy levels to dBm units, do the following:

Energy(dBm) = (energy level – 154)

For example, the energy level reported on channel 0x0B (0x54) is (84 – 154) = -70 dBm.

As a general rule, lower raw energy value readings indicate lower RF energy on the channel. The energy level representation and conversion equations might be different for other (non-Ember) platforms.

API Example 7

Parse the Network Address Response (extended response) received from a device with a 64-bit address of 0x0013A200 404A2257. Use the data in the response to determine the 16-bit address of the device and to determine the addresses of its end device children.

Recall that AO (API Options) must be set on an API device to receive ZDO responses. Suppose the following Explicit Rx API frame is received.

API frame

7E 0022 91 FFFFFFFF FFFFFFFF 0848 00 00 8000 0000 01 01 00 57 22 4A 40 00 A2 13 00 48 08 01 00 AA AC 45

Decoded API frame

0x7E	Start delimiter.
0x0022	Length.
0x91	Explicit receive API frame.
0xFFFFFFFF	FFFFFFFF - 64-bit source address—all 0xFFs if network layer did not include a source 64-bit address. 0x0848 - 16-bit source address.
0x00	Source endpoint—ZDO endpoint.
0x00	Destination endpoint—ZDO endpoint.
0x8000	Cluster ID—Network Address Response.
0x0000	Profile ID—Zigbee Device Profile ID.
0x01	Receive options—packet was acknowledged.

0x010057224A4000A21300480801 00AAAC	Data payload.
0x45	Checksum (0xFF – SUM(all bytes after length).

The data payload bytes can be interpreted into a network address response. Recall that the first byte in the data payload is a transaction sequence number that matches the sequence number of the request.

Data payload bytes (Network Address Response)

0x01	Transaction Sequence Number.
0x00	Status (SUCCESS).
0x57224A40	00A21300 – 64-bit address of the responder—in little-endian byte order. 0x4808 – 16-bit address of the response—in little-endian byte order.
0x01	Number of associated devices—end device children.
0x00	Start index—starting index in the child table list.
0xAAAC	16-bit address of the child—in little-endian byte order.

From the ZDO Network Address Response, we have identified the following:

- The remote with 64-bit address 0x0013A200 404A2257 has a 16-bit address of 0x0848.
- The remote has one end device child.
- The end device child of the remote has a 16-bit address of 0xACAA.

Transmission timeouts

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

The Zigbee stack includes some provisions for a device to detect if the destination is an end device. The Zigbee stack uses the unicast timeout unless it knows the destination is an end device.

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

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

Equations for these timeouts are computed in the following sections.

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

Unicast timeout

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

The unicast timeout includes 3 transmission attempts (1 attempt and 2 retries).

The maximum total timeout is approximately:

3 * ((50 * NH) + 100)

For example, if NH=30 (0x1E), the unicast timeout is approximately 3 * ((50 * 30) + 100) or one of the following:

- 3 * (1500 + 100)
- 3 * (1600)
- 4800 ms
- 4.8 seconds

Extended timeout

The worst-case transmission timeout when you are sending data to an end device is a larger issue than when transmitting to a router or coordinator. As described in Parent operation, RF data packets are sent to the parent of the end device, which buffers the packet until the end device wakes to receive it. The parent buffers an RF data packet for up to (1.2 * **SP**) time.

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

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

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

If no acknowledgment is received within this time, the sender resends the transmission up to two more times. With retries included, the longest transmission timeout when sending data to an end device is:

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

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

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

3 * ((50 * **NH**) + (1.2 * **SP**)) or one of the following:

- 3 * (1500 + 12000)
- 3 * (13500)
- 40500 ms
- 40.5 seconds

Transmission examples

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

API frame

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

Field composition

0x0014	length
0x10	API ID (TX data)
0x01	Frame ID (set greater than 0 to enable the TX-status response)
0x00000000 00000000	64-bit address of coordinator (ZB definition)
0xFFFE	Required 16-bit address if sending data to 64-bit address of 0
0x00	Broadcast radius (0 = max hops)
0x00	Tx options
0x54 78 44 61 74 61	ASCII representation of “TxData” string
0xAB	Checksum (0xFF - SUM (all bytes after length))

Description

This transmission sends the string “TxData” to the coordinator, without knowing the 64-bit address of the coordinator device. ZB firmware defines a 64-bit address of 0 as the coordinator. If the coordinator's 64-bit address was known, the 64-bit address of 0 could be replaced with the coordinator's 64-bit address, and the 16-bit address could be set to 0.

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

API frame

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

Field composition

0x0014	length
0x10	API ID (TX data)
0x01	Frame ID (set to a non-zero value to enable the TX-status response)
0x00000000 0000FFFF	Broadcast definition (including sleeping end devices)
0xFFFE	Required 16-bit address to send broadcast transmission

0x00	Broadcast radius (0 = max hops)
0x00	Tx options
0x54 78 44 61 74 61	ASCII representation of “TxData” string
0xAD	Checksum (0xFF - SUM (all bytes after length))

Description

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

If receiver application addressing is enabled, the XBee 3 Zigbee RF Module reports all received data frames in the explicit format (0x91) to indicate the source and destination endpoints, cluster ID, and profile ID where each packet was received. Status messages like modem status and route record indicators are not affected.

To enable receiver application addressing, set the **AO** command to 1 using the Local AT Command Request - 0x08 as follows:

API frame

7E 0005 08 01 414F 01 65

Field composition

0x0005	length
0x08	API ID (AT command)
0x01	Frame ID (set to a non-zero value to enable AT command response frames)
0x414F	ASCII representation of 'A','O' (the command being issued)
0x01	Parameter value
0x65	Checksum (0xFF - SUM (all bytes after length))

Description

Setting **AO** = **1** is required for the XBee 3 Zigbee RF Module to use the Explicit Receive Indicator - 0x91 when receiving RF data packets. This is required if the application needs indication of source or destination endpoint, cluster ID, or profile ID values used in received Zigbee data packets. ZDO messages can only be received if **AO** = **1**.

Zigbee security

Security overview

Zigbee security protects network traffic using 128-bit AES cryptography techniques. A standard security model is defined for supporting authentication and key management. Security is a very important factor in designing a mesh network. Digi makes it easy to find the right level of security for your specific application, ranging from a completely open and unencrypted network to a high security model with out-of-band device registration.

WARNING! The out-of-the-box default configuration is an unencrypted network with a generous join window. These defaults are meant for ease of development and should not be used on the finished product. Enabling security is highly recommended.

Enabling encryption also enables source routing with the coordinator acting as a high RAM concentrator by default. For smaller networks (less than 40 nodes) and low-throughput applications, this will not have a significant impact to the network, as source routing will automatically be handled by the XBee application. If you are deploying a larger network, you will likely require a full source routing implementation with the coordinator configured as a low RAM concentrator. For more information, see Source routing.

Network key

The network key encrypts and decrypts over the air messages at the network layer. When you enable encryption, each node on the network is required to have the network key to communicate with other nodes. The network key is shared by every device on the network and only needs to be set on the network coordinator. Use the **NK** parameter to set a user-defined network key; this parameter is only applicable to a coordinator (**CE** = **1**). In most situations, the network key should be randomly generated (**NK** = **0**) and managed by the network.

If you are running a centralized trust center, you can change the **NK** parameter on the trust center which propagates to the rest of the network a few seconds later. This is useful for high-security applications where regular network key rotation may be desired. In a distributed trust center, the key is defined when the network is formed and cannot be changed without reforming the network.

Optionally, network keys can be sent and received in-the-clear by setting the **EO** bit 0 (**EO** = **1**) on the forming and joining nodes. Digi strongly discourages this setting, because it could allow unauthorized devices to obtain a copy of the network key.

In addition for centralized trust center you can use RK (Trust Center Network Key Rotation Interval) to do network key rotation (only when **NK** = **0**) with a range of 1 to 22 days automatic interval. Also you can perform a one time key update by setting **RK** to zero, which could be used to extend the time interval beyond 22 days or any interval implemented by your application.

Link key

Link keys are used at the APS layer to provide an extra level of encryption for end-to-end security. The XBee 3 Zigbee application uses global link keys for both joining and APS-encrypted transmissions. When joining a network with encryption enabled, the network key is securely exchanged by encrypting it with the link key.

When using a centralized trust center, the link key that is used to join is exchanged with a more secure key that is randomly generated by the trust center.

This section provides information about the types of link keys.

Preconfigured link key - moderate security

Using a preconfigured global link key provides a very simple way to secure a network, which is accomplished by configuring the same write-only **KY** value on every node on the network. Defining a link key in this manner provides a moderate level of security while allowing for easy network deployment. The security configuration can be done during manufacturing rather than at deployment.

If the joining node has a preconfigured link key that the trust center is not aware of, then it must be registered using an out-of-band method. Issue a 0x24 registration frame on the trust center, which contains the link key and serial number of the joining device.

Well-known default link key - low security

The Zigbee Alliance specifies a well-known default link key. You can use this link key to allow unsecure devices to easily join a secured network. By default, the XBee 3 Zigbee RF Module rejects any device that attempts to join using this well-known key. To allow these devices to join, set the **EO** bit 4 (**EO**=0x10) on the centralized trust center.

If a joining device has **KY** = **0** (default), it attempts to use the well-known default link key to join.

Install code derived link key - high security

Every device supporting Zigbee 3.0 is required to have an install code. Read the install code by querying the **I?** command, which consists of a 16-byte install code + 2 byte CRC. The install code must be read from the joining node and entered to the trust center through an out-of-band method. Typically, the user reads an install code from some type of display or application on the joining node. The user then provides the joiner's install code and serial number to the trust center using a locally issued 0x24 registration API frame by setting bit 0 of the options field.

Using install codes for generating link keys is the most secure method, because it allows users to clearly identify the joining node to the trust center, and it guarantees that each joining device has a random link key.

For a joining device to use an install code, **DC** bit 0 (**DC** = **1**) must be set on the joining device. This generates a link key based on the install code and the **KY** parameter will be ignored.

Join window

Zigbee imposes a limited window of time in which a network can permit joining. The maximum joining window time allowed by the Zigbee specifications is 254 seconds (**NJ** = **0xFE**). Whenever the join window opens, the **NJ** value of the device that opens the window is used. This timeout value is not shared by the rest of the network.

The following conditions cause the network join window to open for **NJ** seconds:

- Local device forms a network (**CE** = **1**).
- A router joins the network. This uses the router's **NJ** value to open the window.
- The commissioning button is enabled (**D0** = **1**) and pressed twice on a router or coordinator on the network.
- A **CB2** command is issued to a router or coordinator on the network.
- A device is successfully registered to the trust center via 0x24 API frame.
- **NJ** parameter value is changed and applied.
- Local device is power cycled.

When the join window opens using **CB2**, the device sends a broadcast to the rest of the network. The joining device does not need to be adjacent to the device that opened the joining window. When setting **NJ** on a device, the join window is only opened on the local device, a broadcast to the network does not get sent.

If **NJ** is set to **0**, the join window remains closed unless explicitly opened via the commissioning button or **CB** command. In this scenario, the join window open for a fixed period of 60 second when opened. For a highly-secured network, Digi recommends setting **NJ** to **0** on every device so the join window does not open inadvertently.

When using an encrypted network with a trust center, opening the join window must be performed on the coordinator. If routers are also needed to allow joining, we recommend executing **CB2** on the coordinator which will broadcast the join window management message thus keeping all devices in sync for both the transient link key and join window timeout. We also recommend setting **KT** and **NJ** to the same value to have a uniform timeout.

Opening the join window will cause the transient link key to be entered into the key table. This allows the link key used by the network to be exchanged with a joining device. If the key table timeout (**KT**) is set lower than **NJ**, a joining device could fail to obtain the necessary keys even though the network allows joining.

When the device executes **CB2** or **NJ** > **0** and **NJ** < **0xFF** the device performs a fast blink while the joining window is open. For all **NJ** settings and executing **CB2** a modem status for opening or closing the join window is sent out the serial port when using **AP1**.

- 0x43: Joining window open
- 0x44: Joining window closed

WARNING! An always-open join window is permitted (**NJ** = **0xFF**), but this causes the network to operate outside of the Zigbee specifications. This option is provided for ease of development and should not be used on the finished product.

Key management

Zigbee defines two security models for key management: centralized security model and distributed security model.

Centralized security

A centralized trust center network is defined as a Zigbee network where one node acts as the centralized key authority. This centralized trust center defines the network key and manages its distribution, determines when and if nodes can join the network, and issues application link keys. Upon formation of the network, the network coordinator assumes the role of the trust center. The trust center has a reserved address of 0 on the network, and any traffic sent to this address is routed to the trust center.

When a node attempts to join, it first establishes a MAC association with a router on the network. The router sends a request to the trust center, indicating the node wants to join. The trust center decides if the node can join based on the current join policy (Open join window + **EO** options). If the trust center approves the attempt to join, the network key is encrypted using a trust center link key and sent to the joining node. The joining node must have a copy of the link key in order to decrypt the network key and successfully join the network.

If the joining node does not have a link key that matches the network or has an install code derived link key, then it must be registered to the trust center. Registration is the means by which a link key is

given to the trust center using an out-of-band method. Registration requires the trust center operate in API mode (AP=1 or 2) and cannot be performed in Command or Transparent mode.

Distributed security

A distributed trust center does not have a node designated as a coordinator. All routers in the network have a copy of the network key and are able to authorize joining devices, meaning every router on the network is a trust center. The network key is set at the time the network is formed and cannot change. The device that forms the network (**CE** is set to **1**) will become the Network Manager. As devices join the network, the Network Manager broadcasts an update with its address information. Any traffic sent to the reserved 0 address will be directed to the Network Manager.

When a node joins a distributed trust center network, an adjacent router shares a copy of the network key to the joining device. The network key is protected by encrypting the exchange with the joining device with a global link key. The network key can optionally be sent in-the-clear by setting **EO** bit 1 on every device on the network. Digi strongly discourages this setting, because it allows unsecure devices access the network key.

You can perform device registration on a distributed trust center, but the 0x24 registration frame must be issued on a router that is adjacent to the joining device; registration information is not shared with the rest of the network.

Device registration

When a device attempts to joins a secure network, it must obtain a copy of the network key to successfully communicate.

You can send the network key in the clear, but in most situations it will be encrypted with a link key. If the link key is not preconfigured on both devices, the trust center must be told the link key the joining device will be using to join. We call this process "registration" and is the method by which a link key and serial number of the joining device is securely given to the trust center through the physical serial interface. Because the registration information is not provided over-the-air, this is considered out-of-band registration and provides the highest level of security since the credentials cannot be extracted through RF channels.

Registration is performed using a Register Joining Device - 0x24 frame and is issued to the trust center (either centralized or distributed). The registration frame is used to register a link key, register an install code derived link key, or remove a previously registered device.

Centralized trust center

On a centralized trust center (**EO** = **2**), registration is transient, meaning that a registered device will only be authorized to join for a fixed period of time. This period is separate from the network join window and is defined by the **KT** parameter on the centralized trust center. By default, a registered device is authorized to join for a period of five minutes. If the device fails to join within this period, it must be re-registered. After joining, it securely rejoins and does not need to be registered again unless the device is explicitly removed from the network using an **NR** command or leave request. The 0x24 registration frame must be issued to the centralized trust center in this scenario, and routers that are adjacent to the joining device route the join request to the trust center. The key table entries on a centralized trust center is stored in RAM and is not preserved across a power cycle.

The key defined by the Trust Center's **KY** parameter value will always persist in the key table and never expire. If **EO** bit 4 is set, then the well-known link key of **ZigbeeAlliance09** will be persistently active in the key table.

Distributed trust center

On a distributed trust center (**EO** = **0**), registration is persistent, meaning that the registered device will always be authorized to join as long as the join window is open. Registration information is not shared to the rest of the network, so the 0x24 registration frame must be issued to a router that is adjacent to the joining device. Because the link key table has a limited number of entries, you must explicitly remove key table entries by deregistering devices using a 0x24 frame after they successfully join to add subsequent devices. The key table on a distributed trust center is stored in flash and persists across a power cycle.

Once a device joins the network and obtains a copy of the network key, it retains information about the network and performs a secure rejoin, if power cycled. If you change a network parameter on the device, it receives a leave request or a secure rejoin fails after three tries. The device must join the network via association which requires registration.

Example: Form a secure network

The following example show how to form a secure Zigbee 3.0 network. This is the recommended configuration for most networks, because it allows for ease of deployment while also maintaining a moderate level of security.

Configure an XBee 3 device with the following parameters:

- **CE** = **1**

  This indicates that the device attempts to form a network rather than join an existing one.

- **EE** = **1**

  This enables encryption for the network.

- **EO** = **2**
  - This forms the network as a centralized trust center. If you want a distributed trust center, set this parameter to 0.
  - Any joining device must have the same value set to properly handle any key exchanges that occur.
- **KY** = non-zero
  - This defines a preconfigured link key for the network.
  - This key can be configured on joining devices as a preconfigured global link key.
  - If joining devices do not use the preconfigured link key, they must be registered to the trust center before joining.
- **NK** = **0**
  - Using a zero **NK** value is preferred, as the XBee will generate a random network key that cannot be read.
  - If acting as a centralized trust center, this parameter can be changed after network formation to update the network key for all devices on the network.
- **NJ** < 0xFF

  This defines the amount of time you want to allow devices to join when the join window opens. You can modify this after the network forms.

If you want to increase the level of security for this network, set **KY** = **0** on the forming node. This generates a random link key that cannot be read and requires every joining device to be individually

registered. This configuration guarantees that only authorized devices can join the network, because the global link key is unclear and cannot be read.

Example: Join a secure network using a preconfigured link key

The following examples show you how join an existing network that has security enabled and the preconfigured link key configured on the network is known. Using this example, it is easy to deploy a secure network, because each device is preconfigured to join the network. An installer only needs to be concerned with opening the join window for new devices.

Configure a joining XBee 3 device with the following parameters:

- **EE** = **1**

  The joining node must have the same encryption settings as the network it will be joining.

- **EO** = **2**
  - If joining a centralized trust center, **EO** bit 1 must be set so the joining device is aware that a link key exchange is needed.
  - If joining a distributed trust center, clear **EO** bit 1.
- **KY** = KY from trust center

  Because the **KY** value is known, it should be preconfigured on the joining device. Provided the **KY** values match, it will be able to obtain the network key and join.

- **NJ** < 0xFF

  Consider the join time that is configured on joining devices. If the device successfully joins the network as a router (**SM** = **0**), it immediately opens the join window for **NJ** seconds, effectively refreshing the window. If you do not wish to reopen the join window in this manner, set **NJ** = **0** on all joining devices.

To join the device to the network, write the previous configuration to flash with a **WR** command, and bring it within RF range of the network.

To open the join window, press the commissioning button twice on a network router or the trust center. If the pushbutton is not available, you can issue a **CB2** command.

Joining devices continuously attempt to join a network (unless explicitly told not to via a **DJ** = **0** command). However, if you want to have the module immediately attempt to join, press the commissioning button once, or issue a **CB1** command on the joining node.

Example: Register a joining node without a preconfigured link key

Using the previous example for joining a network, if the joining node is not aware of the link key on the trust center (that is, it is either obscured (**KY** = **0**) or otherwise unknown to the joining device) then it must be registered to the trust center.

Configure a joining XBee 3 device with the following parameters:

- **EE** = **1**

  The joining node must have the same encryption settings as the network it will be joining.

- **EO** = **2**

- If joining a centralized trust center, **EO** bit 1 must be set so the joining device is aware that a link key exchange is needed.
- If joining a distributed trust center, clear **EO** bit 1.

- **KY** = non-zero value

  Configure a known link key value for this particular joining device. This value must be known by the installer, because it must be passed to the trust center out-of-band.

On the trust center, you must register this device using an API frame. Generate a 0x24 frame that contains the following information:

- The link key (**KY**) of the joining device.
- The serial number of the joining device.

Link Key registration example

A device with the serial number **0013A200 12345678** that has a **KY** of **12345** is trying to join a secure network.

The following 0x24 frame is generated and passed into the UART of the trust center:

7E 00 10 24 7B **00 13 A2 00 12 34 56 78** FF FE **00 01 23 45 31**

The trust center will respond with the following 0xA4 registration response frame:

7E 00 03 A4 7B **00** E0

Note The Frame ID (0x7B) in the response corresponds with the Frame ID of the registration attempt. A 00 result indicates that the key was successfully registered.

When the registration succeeds, the join window automatically opens for **NJ** seconds (or 60 seconds if **NJ** = **0**).

If the trust center is centralized, this registered key table entry is transient and expires after **KT** seconds. In a distributed trust center, it persist until it is explicitly cleared.

Example: Register a joining node using an install code

To provide the highest level of security, Digi recommends using install codes to register devices. Install codes are randomly assigned to each Zigbee 3.0 device at the factory for the purpose of securely joining a network. The process to register a device using an install code is similar to registering a link key, but with some additional steps:

Configure a joining XBee 3 device with the following parameters:

- **EE** = **1**

  The joining node must have the same encryption settings as the network it is joining.

- **EO** = **2**
  - If joining a centralized trust center, **EO** bit 1 must be set so the joining device is aware a link key exchange is needed.
  - If joining a distributed trust center, clear **EO** bit 1.
- **DC** = **1**

  This tells the joining device to generate a link key from the install code of the device. If this bit is enabled, then the device ignores and does not use the **KY** parameter. If you want to register the device with the trust center using the device's link key, do not set the **DC** parameter. The **DC** parameter is only used for registering a device using the **I?** install code.

On the trust center, you must register this device using an API frame. Generate a 0x24 frame that contains the following information:

- The install code (**I?**) of the joining device.
- The serial number of the joining device.

Install code registration example

A device with the serial number **0013A200 12345678** that has a **I?** value of **F6F1913D834A08D6ADAF1F91BAF4052D7316** is trying to join a secure network.

The following 0x24 frame is generated and passed into the UART of the trust center. Set the options field of the API frame to 01 to indicate that the supplied key is actually an install code:

7E 00 1F 24 D5 **00 13 A2 00 12 34 56 78** FF FE **01 F6 F1 91 3D 83 4A 08 D6 AD AF 1F 91 BA F4 05 2D 73 16** 6A

The trust center will respond with the following 0xA4 registration response frame:

7E 00 03 A4 D5 **00** 86

Note The Frame ID (0xD5) in the response corresponds with the Frame ID of the registration attempt. A 00 result indicates that the key was successfully registered.

When the registration succeeds, the join window automatically opens for **NJ** seconds (or 60 seconds if **NJ** = **0**).

If the trust center is centralized, this registered key table entry is transient and expires after **KT** seconds. In a distributed trust center, it persists until explicitly cleared.

Example: Deregister a previously registered device

This feature is only needed in a distributed trust center, because the key table entries are persistent and stored in flash. In a distributed trust center, there are only a limited number of entries available, proper management of the key table is required if more than 10 devices will be joining using registration.

To deregister a device, issue a 0x24 registration frame on the trust center with the serial number of the registered device and a null (blank) key.

Deregistration example

A device with the serial number **0013A200 12345678** that was previously registered has successfully joined the network, and needs to be deregistered to make room for subsequently joining devices.

The following 0x24 frame is generated and passed into the UART of the trust center. Note, that there is no key field, indicating that the key entry should be removed:

7E 00 0D 24 C4 **00 13 A2 00 12 34 56 78** FF FE 00 51

The trust center will respond with the following 0xA4 registration response frame:

7E 00 03 A4 C4 **00** 86

Note The Frame ID (0xC4) in the response corresponds with the Frame ID of the registration attempt. A 00 result indicates that the key was successfully removed from the table.

Registration scenario

It is possible to combine some of the previously mentioned security features to maintain a high level of security with simplified deployment, while also providing a means for authorized devices to

securely join via registration.

For example, an established Zigbee network with a centralized trust center is exhibiting some issues that require analysis by a network engineer. Due to the nature of the deployment, the end user does not want to disclose any of the security credentials to the contracted network engineer.

To allow the network engineer onto the network, the end user must be authorized to join via registration. The network administrator sets the **KT** parameter on the centralized trust center to **0x7080**, which sets the registration timeout to eight hours. Because the network engineer is not yet on-site, the **NJ** parameter can be set to **0xFF** to allow open joining, or opened momentarily via a pressing the commissioning button twice on a router or coordinator when he arrives.

A 0x24 frame is issued to the trust center that contains the serial number of the network engineer's device and a one-time-use link key. The network engineer can then use this link key to join the network and perform whatever work is necessary.

After the analysis has been performed and the network engineer has left the site, the network administrator closes the join window by setting **NJ** to **0**. Deregistration is not needed, because this is a centralized trust center. The temporary link key expires after **KT** seconds, or when the device joins the network through the centralized trust center, the temporary link key will be removed from the table. If the node is removed from the network, it will need to be registered again with the trust center.

Centralized trust center backup

To simplify the recovery of a network that has lost its centralized trust center due to a hardware failure, it is possible to back up the necessary information to restore a centralized trust center using a different physical XBee 3 Zigbee RF Module. This can be done without causing the network to be re-formed in most cases.

Create the backup file

To protect the security key information contained in the backup file from being accessed by unauthorized users, the file is encrypted with 256-bit AES-CTR encryption. Use KB (Centralized Trust Center Backup Key) to set the encryption key; it must be set before a backup file can be created. You also need to set **KB** to the same value on the new device prior to restoring.

New networks

During the initial configuration of a centralized trust center, the backup encryption key should be set using the **KB** command. Once a centralized trust center has been configured and formed a network, an encrypted backup file can be created using BK (Centralized Trust Center Backup and Restore).

Existing networks

It is possible to begin backing up an existing centralized trust center without reforming its network if a preconfigured link key was previously set on the trust center—**KY** value is non-zero.

To ensure security, the backup encryption key—**KB**—must be set prior to creating the backup file. To set **KB** while maintaining the current key information—**KY** and **NK**—the current value of **KY** must be provided to verify that **KB** is being set by an authorized administrator of the network. If **KB** is set without providing **KY**, the existing values of **KY** and **NK** are cleared; you must take care, as this effectively invalidates the current network. See KB (Centralized Trust Center Backup Key) for more details.

Note Once **KB** has been set the first time, then the current **KB** key can be used to set a new key in place of **KY**. See KB (Centralized Trust Center Backup Key) for more details.

Store the file

Once a backup file has been created, it will be located in the device's file system with the name **backup_TC<SL>.xbee** where **SL** is the lower 32 bits of the device address. This file can be retrieved from the trust center using XCTU's File System Manager. The backup file must be stored off of the device as it may be impossible to retrieve after a hardware failure. Ideally, a new backup file should be periodically created and retrieved. If NWK keys are regularly rotated, we advise saving a backup after the rotation.

The backup file and value of **KB** should be kept secure and appropriately safeguarded against access by unauthorized users.

Recover a Centralized Trust Center

In the event of a hardware failure, the failed device may be replaced with a new device. The backup file can be transferred to the new devicee using the File System Manager. You must set the same encryption key value using the **KB** command and the restore operation can be completed with the **BK** command.

After restoring a trust center backup, the configuration of the new device will completely match that of the previous trust center. If the backup file is sufficiently recent, the coordinator should now be able to communicate with the existing network and the process is complete.

If the restored coordinator is not able to communicate with the network it is possible that some of the network information contained in the file may be out of date. You can retrieve updated network information from a router that is still on the network using the **CX** command. The output of this command is then used as additional parameters for the **BK** command to update the new

coordinator's network information. Finally, the network should be configured with a new network key by issuing a **NK** with a new key value to the replacement trust center.

Best practices

We recommend setting a value for **KY** even when using install codes. This allows **KB** to be set on a deployed network using the **KY** value without network disruption.

After a restore, no attempt should be made to reuse the original device that was used to create the backup. This is because the new device's EUI64 (**SH** + **SL**) will have been permanently changed to that of the original device and an attempt to use both could result in networking conflicts. This procedure is intended for situations where the original XBee 3 device has been rendered inoperable.

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.

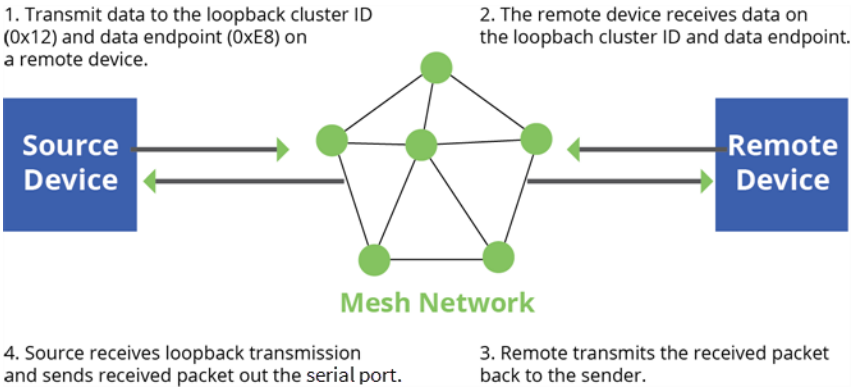
Place devices

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

Test links in a network - loopback cluster

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

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



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

Transparent operating mode configuration (AP = 0)

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

1. Set the **CI** command to **0x12**.
2. Set the **SE** and **DE** commands to **0xE8** (default value).

3. Set the **DH** and **DL** commands to the address of the remote (**0** for the coordinator, or the 64-bit address of the remote).

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

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

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

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

RSSI indicators

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

Determine the **DB** value in hardware using the RSSI/PWM device pin (Micro pin 7/SMT pin 7/TH pin 6). If you enable the RSSI PWM functionality (**P0** command), when the device receives data, it sets the RSSI PWM to a value based on the RSSI of the received packet (this value only indicates the quality of the last hop). You could connect this pin to an LED to indicate if the link is stable or not.

Device discovery

Network discovery

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

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

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

ZDO discovery

The Zigbee device profile includes provisions to discover devices in a network that are supported on all Zigbee devices (including non-Digi products). These include the LQI Request (cluster ID 0x0031) and the Network Update Request (cluster ID 0x0038). You can use the LQI Request to read the devices in the neighbor table of a remote device, and the Network Update Request for a remote device to complete an active scan to discover all nearby Zigbee devices. You can send both of these ZDO commands using the Explicit Addressing Command Request - 0x11. For more information, see API Operation. Refer to the Zigbee specification for formatting details of these two ZDO frames.

Joining Announce

All Zigbee devices send a ZDO Device Announce broadcast transmission when they join a Zigbee network (ZDO cluster ID 0x0013). These frames are sent out the device's serial port as an Explicit Rx Indicator frame - 0x91 if **AO** is set to **1**.

The device announce payload includes the following information:

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

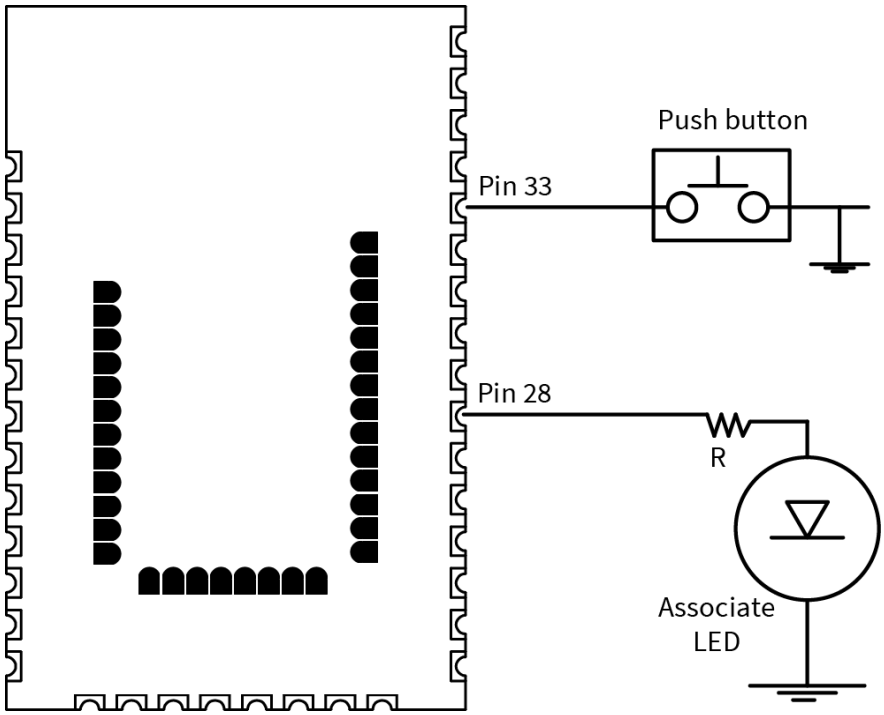
The 16-bit and 64-bit addresses are received in little-endian byte order (LSB first). See the Zigbee specification for details.

Any received Network Address Requests (ZDO cluster ID 0x0000) and IEEE Address Request (ZDO Cluster ID 0x0001) will also be emitted if **AO** is set to **1**.

Commissioning pushbutton and associate LED

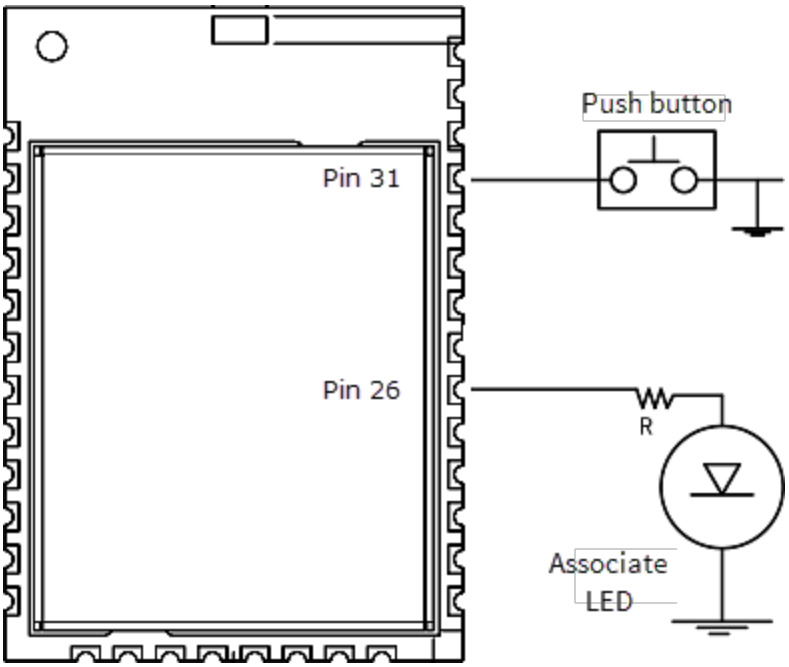
XBee devices support a set of commissioning pushbutton and LED behaviors to aid in device deployment and commissioning. These include the commissioning push button definitions and associate LED behaviors. The following features can be supported in hardware:

XBee 3 SMT



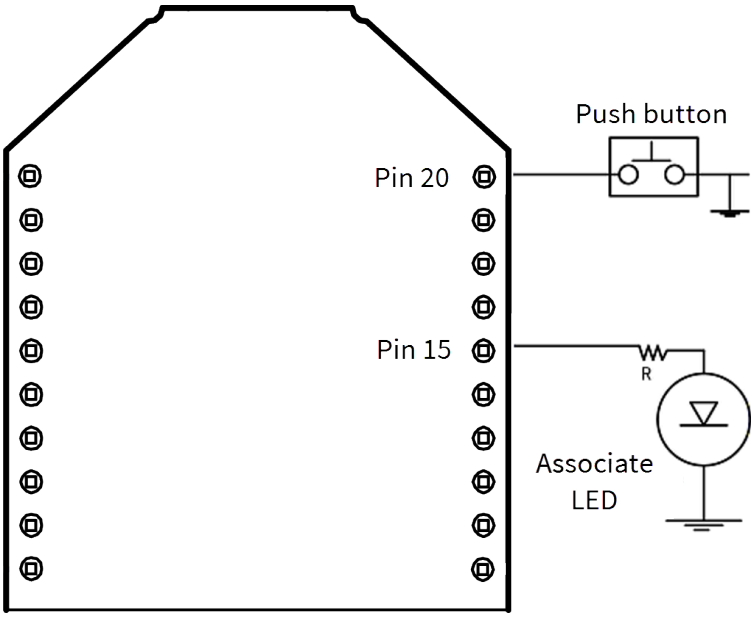
A pushbutton and an LED can be connected to the surface-mount device to support the commissioning pushbutton and associate LED functionalities.

XBee 3 Micro



A pushbutton and an LED can be connected to the Micro device to support the commissioning pushbutton and associate LED functionalities.

XBee 3 Through-hole



A pushbutton and an LED can be connected to the through-hole-mount device to support the commissioning pushbutton and associate LED functionalities.

Commissioning pushbutton

The commissioning pushbutton definitions provide a variety of simple functions to help with deploying devices in a network. Enable the commissioning button functionality by setting D0 (DIO0/AD0/Commissioning Button Configuration) to **1** (enabled by default).

Button presses	Description
1	Start Joining. Wakes a sleeping end device for 30 seconds, regardless of the **ST**/**SN** setting. It also sends node identification broadcast if joined to a network. A Zigbee device blinks a numeric error code on the Associate pin indicating the cause of join failure for (**AI** - 32) times. A **SE** router or **SE** end device which is associated but not authenticated to a network leaves its network; then attempt to join.
2	Enable Joining. Broadcast a Mgmt_Permit_Joining_req (ZDO ClusterID 0x0036) with TC_ Significance set to **0x00**. If **NJ** is **0x00** or **0xFF**, PermitDuration is set to one minute, otherwise PermitDuration is set to **NJ**.
4	Restore configuration to default values and leave the network. Equivalent to issuing **NR**, **RE**, and **AC** commands.

Use CB (Commissioning Pushbutton) to simulate button presses in software. Issue a **CB** command with a parameter set to the number of button presses you want executed. For example, sending **CB1** executes the actions associated with a single button press.

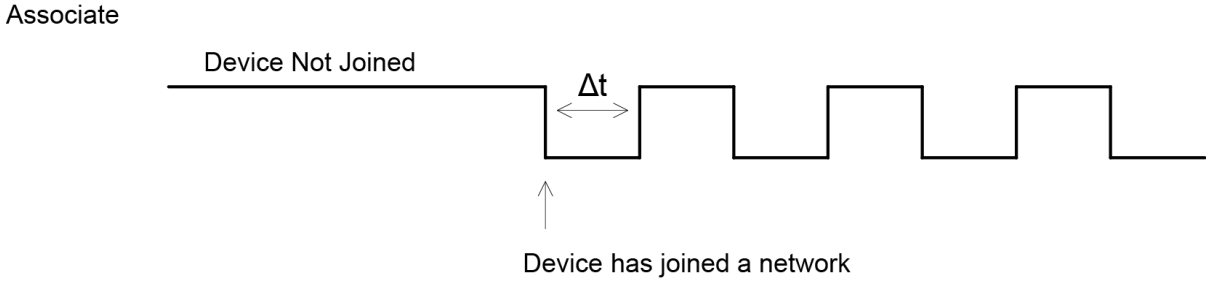
The node identification frame is similar to the node discovery response frame; it contains the device's address, node identifier string (**NI** command), and other relevant data. All API devices that receive the node identification frame send it out their serial interface as a Node Identification Indicator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

Associate LED

The Associate pin provides an indication of the device's network status and diagnostics information. Connect an LED to the Associate pin as shown in the figure in Commissioning pushbutton and associate LED. Enable the Associate LED functionality the **D5** command to 1 (enabled by default). If the Associate pin is enabled, it configured as an output.

Joined indication

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



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

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

Open Join Window indication

The Associate pin indicates when the Network Join Window is open. If the device is allowing joining, the association led will blink at 100 ms.

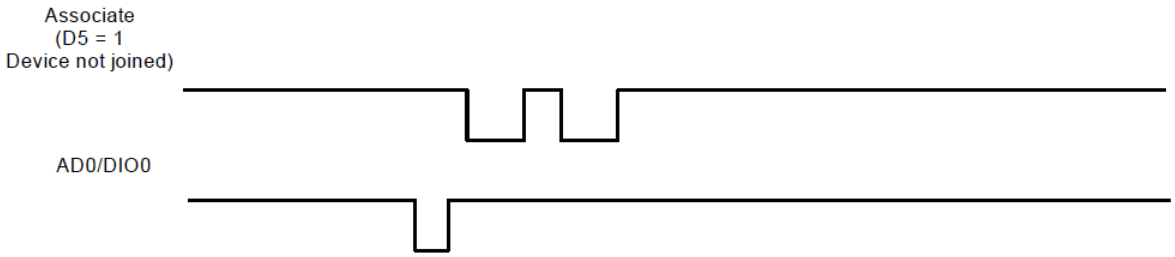
See Join window for information on the join window and what circumstances can cause it to open.

Diagnostics support

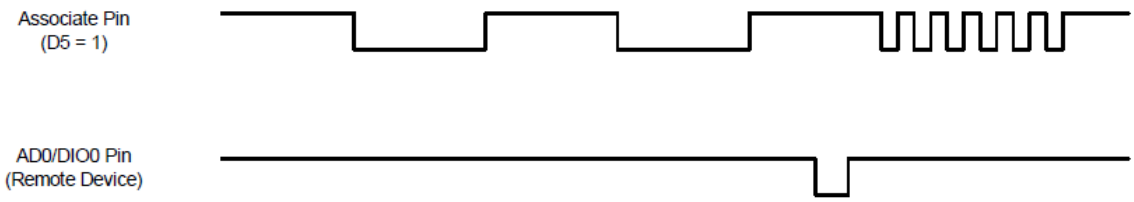
The Associate pin works with the commissioning pushbutton to provide additional diagnostics behaviors to aid in deploying and testing a network. If the commissioning push button is pressed once, and the device has not joined a network, the Associate pin blinks a numeric error code to indicate the cause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

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

The following image illustrates the behavior pressing the commissioning button press once when the device has not joined a network, causing the associate pin to blink to indicate the **AI** Code where: **AI** = # blinks + 0x20. In this example, **AI** = **0x22**.



The following image illustrates the behavior pressing the button once on a remote device, causing a broadcast node identification transmission to be sent. All devices that receive this transmission blink their associate pin rapidly for one second if the associate LED functionality is enabled (**D5** = **1**).



Binding

The Digi XBee firmware supports three binding request messages:

- End Device Bind
- Bind
- Unbind

End_Device_Bind_req

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

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

The coordinator and other devices being setup for binding should have **AO** set to **3**.

R1->C End_Device_Bind_req

R2->C End_Device_Bind_req

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

C matches the requests.

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

R2<-C Unbind_req

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

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

R1<-C Bind_req

R1->C Bind_rsp

R2<-C Bind_req

R2->C Bind_rsp

C sends responses to the original End_Device_Bind_req messages.

R1-<C End_Device_Bind_rsp

R2-<C End_Device_Bind_rsp

End Device binding sequence (binding)

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

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

R1->C End_Device_Bind_req

R2->C End_Device_Bind_req

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

C matches the requests.

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

R2<-C Unbind_req

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

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

R1<-C Unbind_req

R1->C Unbind_rsp

R2<-C Unbind_req

R2->C Unbind_rsp

C sends responses to the original End_Device_Bind_req messages.

R1-<C End_Device_Bind_rsp

R2-<C End_Device_Bind_rsp

Example of an End_Device_Bind_req

This example shows a correctly formatted End_Device_Bind_req (ZDO cluster 0x0020) using a Digi 0x11 Explicit API Frame. The coordinator and other devices being setup for binding should have **AO** set to **3** and must send all binding requests within the 60 second binding timeout period.

The frame as a bytelist

7E 00 28 11 01 00 00 00 00 00 00 00 00 FF FE 00 00 00 20 00 00 00 00 03 3B 05 15 95 75 01 FF A2 13 00 D5 05 C1 01 01 00 01 02 00 19

Same frame broken into labeled fields

Note Multibyte fields are represented in big-endian format.

7e	Frame Delimiter
0028	Frame Length
11	API Frame Type (Explicit Frame)
01	Frame Identifier (for response matching)
0000000000000000	Coordinator address
fffe	Code for unknown network address
00	Source Endpoint (need not be 0x00)
00	Destination Endpoint (ZDO endpoint)
0020	Cluster 0x0020 (End_Device_Bind_req)
0000	ProfileID (ZDO)
00	Radius (default, maximum hops)
00	Transmit Options
03 3B 05 15 95 75 01 FF A2 13 00 D5 05 C1 01 01 00 01 02 00	RFData (ZDO payload)
19	Checksum

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

01	Transaction Sequence Number
3B 05	Binding Target (16 bit network address of sending device)
15 95 75 01 FF A2 13 00	(64 bit address of sending device)

D5	Source Endpoint on sending device
05c1	ProfileID (0xC105) - used when matching End_Device_Bind_requests
01	Number of input clusters
0100	Input cluster ID list (0x0100)
01	Number of output clusters
0200	Output cluster ID list (0x0200)

Bind_req

The Bind request (ZDO cluster 0x0021) is described in the Zigbee Specification. A binding may be coded for either a unicast or a multicast/groupID message.

Example of an Unbind Req

This example shows a correctly formatted Unbind_req (ZDO cluster 0x0022) using a Explicit Addressing Command Request - 0x11 sent by the coordinator.

The frame as a bytelist

7E 00 2A 11 01 00 13 A2 FF 01 75 95 15 FF FE 00 00 00 22 00 00 00 00 03 15 95 75 01 FF A2 13 00 D5 01 00 03 22 96 08 71 FF A2 13 00 D5 90

Same frame broken into labeled fields

Note Multibyte fields are represented in big-endian format.

7e	Frame Delimiter
0028	Frame Length
11	API Frame Type (Explicit Frame)
01	Frame Identifier (for response matching)
00 13 A2 FF 01 75 95 15	Device address which has binding entry
fffe	Code for unknown network address
00	Source Endpoint (need not be 0x00)
00	Destination Endpoint (ZDO endpoint)
0022	Cluster 0x0022 (Unbind_req)
0000	ProfileID (ZDO)
00	Radius (default, maximum hops)
00	Transmit Options

03 15 95 75 01 FF A2 13 00 D5 01 00 03 22 96 08 71 FF A2 13 00 D5	RFData (ZDO payload)
90	Checksum

Here is the RFData (the ZDO payload) broken into labeled fields.

Note The multi-byte fields of a ZDO payload are represented in little-endian format.

03	Transaction Sequence Number
15 95 75 01 FF A2 13 00	64 bit address of the source address used in the R1 end device binding request
D5	Source Endpoint used in the R1 end device binding request
01 00	Output cluster ID used in the R1 end device binding request
03	Fixed value for indicating destination address mode includes endpoint instead of group address
22 96 08 71 FF A2 13 00	64 bit address of the source address used in the R2 end device binding request
D5	Destination Endpoint used in the R1 end device binding request

Group Table API

Unlike the Binding Table that is managed with ZDO commands, a Zigbee group table is managed by the Zigbee cluster library Groups Cluster (0x0006) with ZCL commands.

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

But to implement multicast (group) addressing within the XBee, the external processor must keep the XBee device's group table state in sync with its own. For this reason, a Group Table API has been defined where the external processor can manage the state of the XBee 3 Zigbee RF Module group table.

The design of the Group Table API of the XBee firmware derives from the ZCL Group Cluster 0x0006. Use the Explicit Addressing Command Request - 0x11 addressed to the Digi Device Object endpoint (0xE6) with the Digi XBee ProfileID (0xC105) to send commands and requests to the local device.

The Zigbee home automation public application profile says groups should only be used for sets of more than five devices. This implies sets of five or fewer devices should be managed with multiple binding table entries.

There are five commands implemented in the API:

- Add Group command
- View group
- Remove Group
- Remove All Groups

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

Add Group command

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

The following example adds a group table entry which associates endpoint E7 with groupID 1234 and groupName “ABCD”.

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

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

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

ZCL_header = “01 ee 00"

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

ZCL_payload = “3412 04 41 42 43 44"

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

The example packet in raw hex byte form:

7e001e11010013a2004047b55cfffee6e70006c105000001ee0034120441424344c7

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

7e0018910013a2004047b55cfffee7e68006c1050009ee0000341238

7e00078b01fffe00000076

The response in decoded form:

Zigbee Explicit Rx Indicator

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

ClusterID 0x8006 ProfileID 0xC105 Options 0x00

RF_Data 0x09EE00003412

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

Preamble = “910013a2004047b55cfffee7e68006c10500”

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

ZCL_header = “09 ee 00"

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

ZCL_payload = “00 3412"

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

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

Zigbee Tx Status

API 0x8B FrameID 0x01 16DestAddr 0xFFFE

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

View group

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

The following example gets the name string associated with the endpoint E7 and groupID 1234.

The packet:

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

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

```
ZCL_header = “01 ee 01"
```

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

```
ZCL_payload = “3412”
```

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

The packet in raw hex byte form:

```
7e001911010013a2004047b55cfffee6e70006c105000001ee013412d4
```

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

```
7e001d910013a2004047b55cfffee7e68006c1050009ee01003412044142434424
7e00078b01fffe00000076
```

The command response in decoded form:

```
Zigbee Explicit Rx Indicator
API 0x91 64DestAddr 0x0013A2004047B55C 16DestAddr 0xFFFE SrcEP 0xE7 DestEP 0xE6
ClusterID 0x8006 ProfileID 0xC105 Options 0x00
RF_Data 0x09EE010034120441424344
```

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

```
Preamble = “910013a2004047b55cfffee7e68006c10500”
```

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

```
ZCL_header = “09 ee 01"
```

The first field is a frame control field which specifies a Cluster Specific command (0x01) using a Server->Client direction (0x08). The second field is a transaction sequence number which associates the response with the command request. The third field is the command identifier “View Group” (0x01) .

```
ZCL_payload = “00 3412 0441424344"
```

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

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

```
Zigbee Tx Status
API 0x8B FrameID 0x01 16DestAddr 0xFFF
Transmit Retries 0x00 Delivery Status 0x00 Discovery Status 0x00 Success
```

Get Group Membership (1 of 2)

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

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

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

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

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

```
ZCL_header = “01 ee 02"
```

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

```
ZCL_payload = “00”
```

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

The example packet in raw hex byte form:

```
7e001811010013a2004047b55cfffee6e70006c105000001ee020019
```

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

```
7e0019910013a2004047b55cfffee7e68006c1050009ee02ff01341235
```

```
7e00078b01fffe00000076
```

The response in decoded form:

```
Zigbee Explicit Rx Indicator
API 0x91 64DestAddr 0x0013A2004047B55C 16DestAddr 0xFFFE SrcEP 0xE7 DestEP 0xE6
ClusterID 0x8006 ProfileID 0xC105 Options 0x00
RF_Data 0x09EE02FF013412
```

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

```
Preamble = “910013a2004047b55cfffee7e68006c10500”
```

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

```
ZCL_header = “09 ee 02"
```

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

```
ZCL_payload = “FF 01 3412"
```

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

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

```
Zigbee Tx Status
API 0x8B FrameID 0x01 16DestAddr 0xFFFE
Transmit Retries 0x00 Delivery Status 0x00 Discovery Status 0x00 Success
```

Get Group Membership (2 of 2)

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

The following example gets the groupIDs associated with endpoint E7 which are a subset of a given list of groupIDs (0x1234, 0x5678).

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

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

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

```
ZCL_header = “01 ee 02"
```

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

```
ZCL_payload = “02 34127856"
```

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

The example packet in raw hex byte form:

```
 7e001c11010013a2004047b55cfffee6e70006c105000001ee02023412785603
```

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

```
7e0019910013a2004047b55cfffee7e68006c1050009ee02ff01341235
7e00078b01fffe00000076
```

The response in decoded form:

```
Zigbee Explicit Rx Indicator
```

```
API 0x91 64DestAddr 0x0013A2004047B55C 16DestAddr 0xFFFE        SrcEP 0xE7    DestEP
0xE6
ClusterID 0x8006 ProfileID 0xC105    Options 0x00
RF_Data 0x09EE02FF013412
```

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

```
Preamble = “910013a2004047b55cfffee7e68006c10500”
```

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

```
ZCL_header = “09 ee 02"
```

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

```
ZCL_payload = “FF 01 3412"
```

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

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

```
Zigbee Tx Status
API 0x8B       FrameID 0x01    16DestAddr 0xFFFE
Transmit Retries 0x00 Delivery Status 0x00 Discovery Status 0x00 Success
```

Remove Group

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

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

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

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

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

```
ZCL_header = “01 ee 03"
```

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

```
ZCL_payload = “3412”
```

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

The packet in raw hex byte form:

```
7e001911010013a2004047b55cfffee6e70006c105000001ee033412d2
```

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

```
 7e0018910013a2004047b55cfffee7e68006c1050009ee0300341235
7e00078b01fffe00000076
```

The command response in decoded form:

```
Zigbee Explicit Rx Indicator
API 0x91 64DestAddr 0x0013A2004047B55C 16DestAddr 0xFFFE       SrcE 0xE DestEP 0xE6
ClusterID 0x8006 ProfileID 0xC105    Options 0x00
RF_Data 0x09EE03003412
```

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

```
 Preamble = “910013a2004047b55cfffee7e68006c10500”
```

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

```
ZCL_header = “09 ee 03"
```

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

```
ZCL_payload = “00 3412"
```

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

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

```
Zigbee Tx Status
API 0x8B        FrameID 0x01    16DestAddr 0xFFFE
Transmit Retries 0x00 Delivery Status 0x00 Discovery Status 0x00 Success
```

Remove All Groups

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

The following example removes all groups associated with endpoint E7.

The packet:

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

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

```
ZCL_header = “01 ee 04"
```

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

```
ZCL_payload = “”
```

No payload is needed for this command.

The packet in raw hex byte form:

```
7e001711010013a2004047b55cfffee6e70006c105000001ee0417
```

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

```
7e0016910013a2004047b55cfffee7e68006c1050009ee04007c
7e00078b01fffe00000076
```

The command response in decoded form:

```
Zigbee Explicit Rx Indicator
API 0x91 64DestAddr 0x0013A2004047B55C 16DestAddr 0xFFFE       SrcEP 0xE7    DestEP
0xE6
ClusterID 0x8006 ProfileID 0xC105    Options 0x00
RF_Data 0x09ee0400
```

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

```
Preamble = “910013a2004047b55cfffee7e68006c10500”
```

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

```
ZCL_header = “09 ee 04"
```

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

```
ZCL_payload = “00”
```

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

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

```
Zigbee Tx Status
API 0x8B       FrameID 0x01    16DestAddr 0xFFFE
Transmit Retries 0x00 Delivery Status 0x00 Discovery Status 0x00 Success
```

Default responses

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

```
ZCL_header = “08 01 03" - general command/server-to-client, transseqnum=1,
default_response_command(0x03)
ZCL_payload = “78 8b” - original cmdID, status code (0x8b) status not found
```

Common status codes

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

```
0x00: Command request was successful
0x01: Command request failed - for example,
```

```
a call to remove an entry from the group table returned an error
0x80: no RFData in the API frame;
ZCL Payload appears truncated from what is expected
0x81: unexpected direction
in the Frame Control Field of the ZCL Header; unexpected command identifier
code value
in the ZCL header
0x82: unexpected frametype
in the Frame Control Field of the ZCL Header
0x84: unexpected
manufacturer specific indication in the Frame Control Field of the ZCL Header
0x8b: An attempt at Get Group Membership or
Remove Group could not find a matching entry in the group table
```

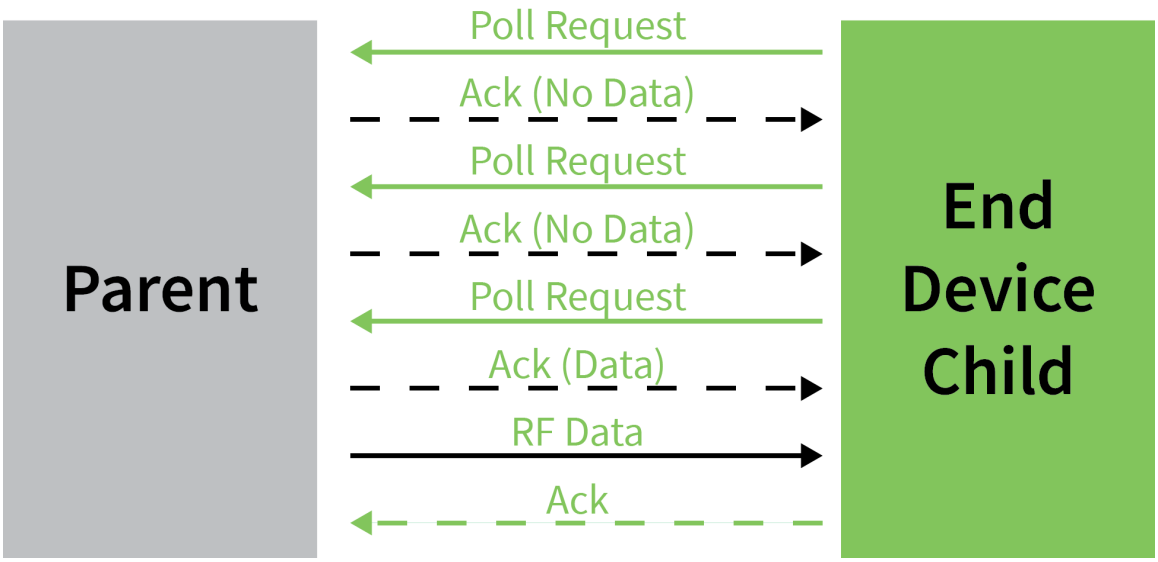
Manage End Devices

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

End device operation

When an end device joins a Zigbee network, it must find a router or coordinator device that is allowing end devices to join. Once the end device joins a network, it forms a parent-child relationship with the end device and the router or coordinator that allowed it to join. For more information, see Zigbee networks.

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



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

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

The parent of the receiving device does not send the network ACK back to the originator until the sleeping end device wakes and polls the data or until the timeout occurs.

Parent operation

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

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

The parent can buffer one broadcast transmission for all of its end device children. When the parent receives and buffers a broadcast transmission, it sets a flag in its child table when each child polls and retrieves the packet. Once all children have received the broadcast packet, the parent discards the buffered broadcast packet. If all children have not received a buffered broadcast packet and the parent receives a new broadcast, it discards the old broadcast packet, clears the child table flags, and buffers the new broadcast packet for the end device children as shown in the following figure.

End Device Child Table

Address	Received Broadcast
Ox2120	T
OxF220	F
OxC100	F
Ox5750	T

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

End Device poll timeouts

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

End Device child table

The child table timeout is controlled by setting the ET (End Device Timeout) on the End Device. The End Device sends the child table timeout value to the parent when joining a network. The **ET** setting should be a value greater than the expected End Device sleep time—see ET (End Device Timeout) for child table timeout values.

Packet buffer usage

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

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

Applications that use regular broadcasting or that require regular address or route discoveries use up a significant number of buffers, reducing the buffer availability for managing packets for end device

children. Applications can reduce the number of required application broadcasts, and consider implementing an external address table or many-to-one and source routing if necessary to improve routing efficiency.

Non-Parent device operation

Devices in the Zigbee network treat data transmissions to end devices differently than transmissions to other routers and coordinators. When a device sends a unicast transmission, if it does not receive a network acknowledgment within a timeout, the device resends the transmission. When transmitting data to remote coordinator or router devices, the transmission timeout is relatively short since these devices are powered and responsive.

However, since end devices may sleep for some time, unicast transmissions to end devices use an extended timeout mechanism in order to allow enough time for the end device to wake and receive the data transmission from its parent.

If a non-parent device does not know the destination is an end device, it uses the standard unicast timeout for the transmission. However, provisions exist in the Silicon Labs Zigbee stack for the parent to inform the message sender that the destination is an end device. Once the sender discovers the destination device is an end device, future transmissions will use the extended timeout. For more information see Router/Coordinator configuration.

End Device configuration

XBee end devices support four different sleep modes:

- Pin sleep
- Cyclic sleep
- Cyclic sleep with pin wake-up
- MicroPython sleep (with optional pin wake). For complete details see the *Digi MicroPython Programming Guide*.

Pin sleep allows an external microcontroller to determine when the XBee 3 Zigbee RF Module sleeps and when it wakes by controlling the SLEEP_RQ pin. In contrast, cyclic sleep allows the sleep period and wake times to be configured through the use of AT commands. Cyclic sleep with pin wake-up is the same as cyclic sleep except the device can be awakened before the sleep period expires by lowering the SLEEP_RQ line. The **SM** command configures the sleep mode. The end device continues to stay awake as long as DTR is held low. The device resumes its sleeping pattern upon driving DTR high again.

In both pin and cyclic sleep modes, XBee end devices poll their parent every 100 ms while they are awake to retrieve buffered data. When the end device sends a poll request, it enables the receiver until it receives an acknowledgment from the parent. It typically takes less than 10 ms between sending the poll request to receiving the acknowledgment. The acknowledgment indicates if the parent has buffered data for the end device child. If the acknowledgment indicates the parent has pending data, the end device leaves the receiver on to receive the data. Otherwise, the end device turns off the receiver and enter idle mode (until it sends the next poll request) to reduce current consumption (and improve battery life).

Once the device enters sleep mode, the On/Sleep pin (Micro pin 25/SMT pin 26) it de-asserts (low) to indicate the device is entering sleep mode. If the device enables CTS hardware flow control (**D7** command), it de-asserts (high) the CTS pin (Micro pin 24/SMT pin 25) when entering sleep to indicate that serial data should not be sent to the device.

If the Associate LED pin is configured (**D5** command), the associate pin is driven low to avoid using power to light the LED. The SLEEP_RQ pin is configured as a pulled-down input so that an external

device must drive it low to wake the device. All other pins are left unmodified during sleep so that they can operate as previously configured by the user. The device does not respond to serial or RF data when it is sleeping.

Applications that must communicate serially to sleeping end devices are encouraged to observe CTS flow control.

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

Pin sleep

Pin sleep allows the module to sleep and wake according to the state of the DTR/SLEEP_RQ pin (Micro pin 9/SMT pin 10/TH pin 9). Pin sleep mode is enabled by setting the **SM** command to 1.

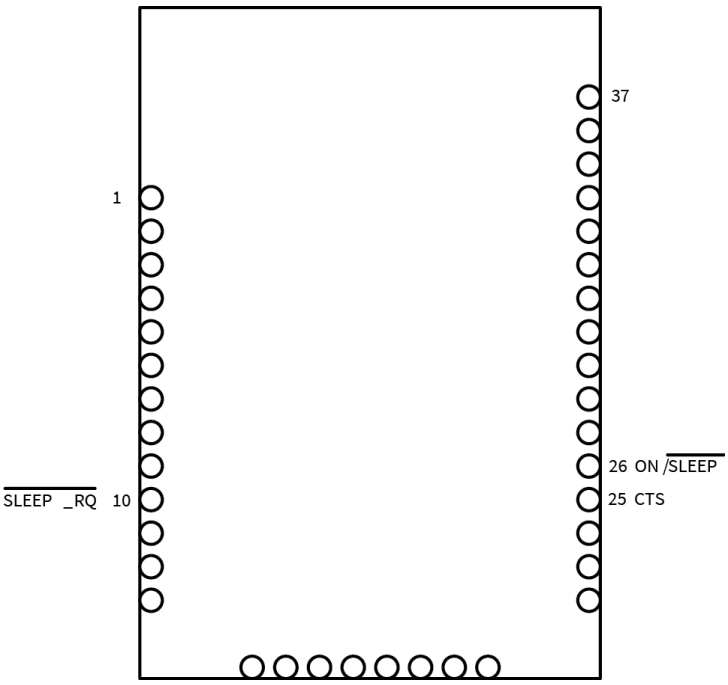
When the device asserts (high) DTR/SLEEP_RQ, it finishes any transmit or receive operations for the current packet that is processing and enters a low power state. For example, if the device has not joined a network and SLEEP_RQ is asserted (high), it sleeps once the current join attempt completes (that is, when scanning for a valid network completes). The device wakes from pin sleep when the SLEEP_RQ pin is de-asserted (low).

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 (set **P5** , **P6** , and **P8** to **0**). See Low power operation for information on using SPI_SSEL for sleep control while communicating over SPI.

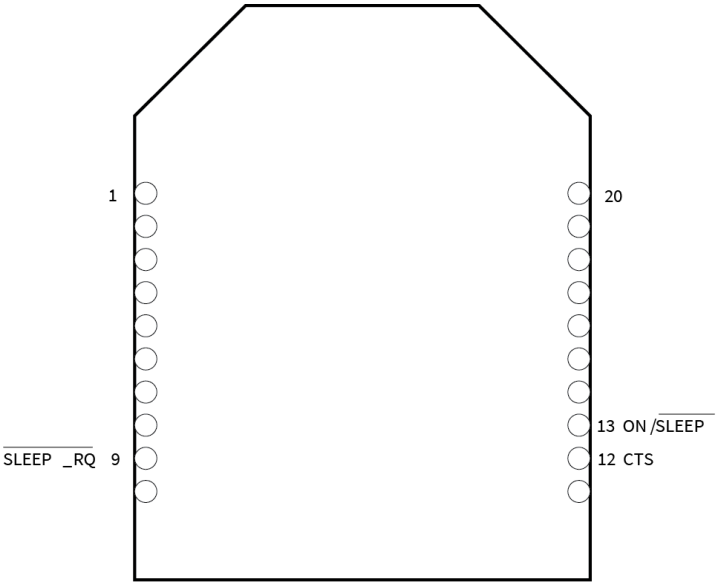
Sleep pin diagrams

The following figures show the device's sleep pins.

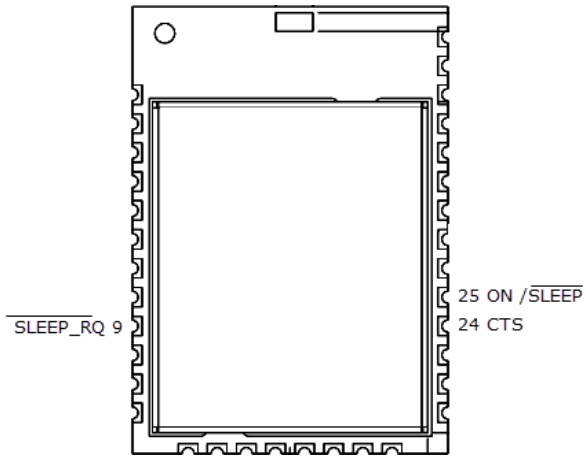
Surface-mount sleep pins



Through-hole sleep pins

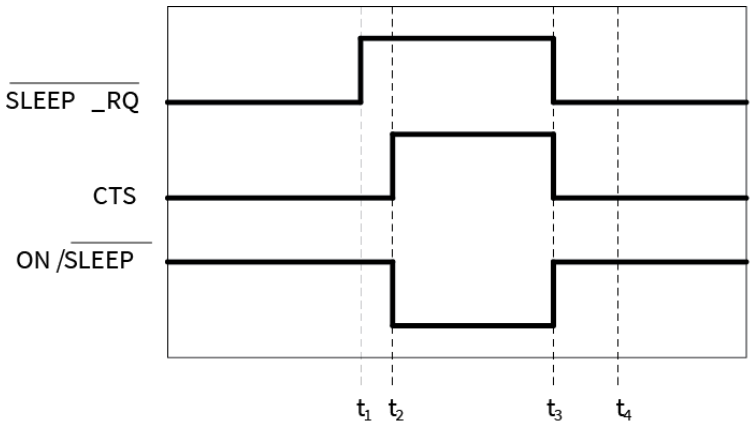


Micro sleep pins



Sleep pin waveform

The following figure show the pin sleep waveforms:



In the previous figure, t1, t2, t3 and t4 represent the following events:

- t1 - Time when DTR/SLEEP_RQ is asserted (high)
- t2 - Time when the device enters sleep (CTS state change only if hardware flow control is enabled)
- t3 - Time when DTR/SLEEP_RQ is de-asserted (low) and the device wakes
- t4 - Time when the device sends a poll request to its parent

The time between t1 and t2 varies depending on the state of the module. In the worst case scenario, if the end device is trying to join a network, or if it is waiting for an acknowledgment from a data transmission, the delay could be up to a few seconds. The time between t3 and t4 is 1-2 ms for a regular device and about 6 ms for a PRO device.

When the XBee 3 Zigbee RF Module is awake and is joined to a network, it sends a poll request to its parent to see if the parent has any buffered data. The end device continues to send poll requests every 100 ms while it is awake.

Demonstration of pin sleep

Parent and remote devices must be configured to buffer data correctly and to use adequate transmission timeouts. For more information, see Router/Coordinator configuration.

Cyclic sleep

Cyclic sleep allows the device to sleep for a specified time and wake for a short time to poll its parent for any buffered data messages before returning to sleep again. Enable cyclic sleep mode by setting the **SM** command to 4 or 5. **SM**5 is a slight variation of **SM**4 that allows the device to wake up prematurely by asserting (low) the DTR/SLEEP_RQ pin. In **SM**5, the XBee 3 Zigbee RF Module can wake after the sleep period expires, or if a high-to- low transition occurs on the SLEEP_RQ pin. When the device wakes due to DTR/SLEEP_RQ being asserted (low), the minimum time that it will wake for is ST (Cyclic Sleep Wake Time) even if DTR/SLEEP_RQ is again de-asserted sooner. Setting **SM** to 4 disables the pin wake option.

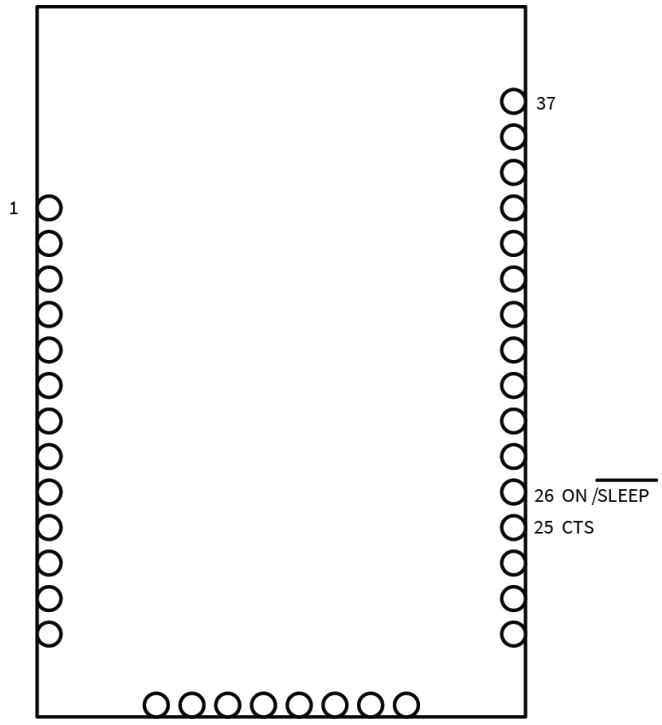
In cyclic sleep, the device sleeps for a specified time, and then wakes and sends a poll request to its parent to discover if the parent has any pending data for the end device. If the parent has buffered data for the end device, or if it receives serial data, the device remains awake for a time. Otherwise, it enters sleep mode immediately.

When the device wakes, it asserts (high) the ON/SLEEP line, and de-asserted (low) when the device sleeps. If you enable hardware flow control (**D7** command), the CTS pin asserts (low) when the device wakes and can receive serial data, and de-assert (high) when the device sleeps.

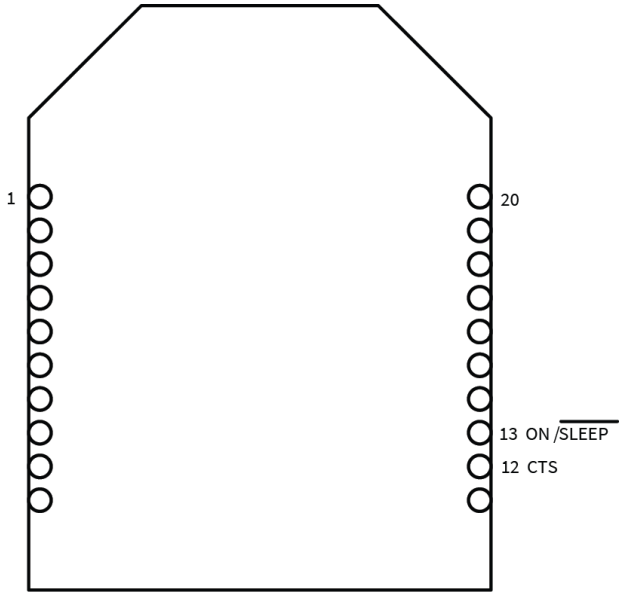
Cyclic sleep pin diagrams

The following figure shows the device's cyclic sleep pins.

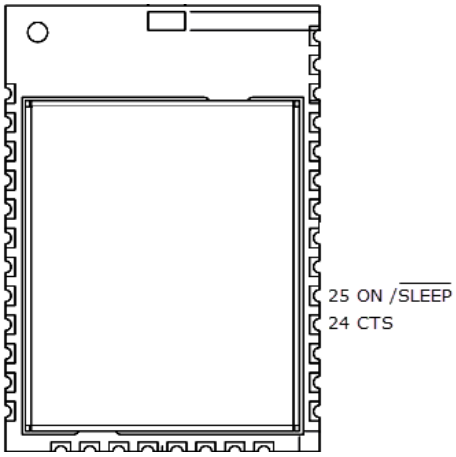
Surface-mount cyclic sleep pins



Through-hole cyclic sleep pins

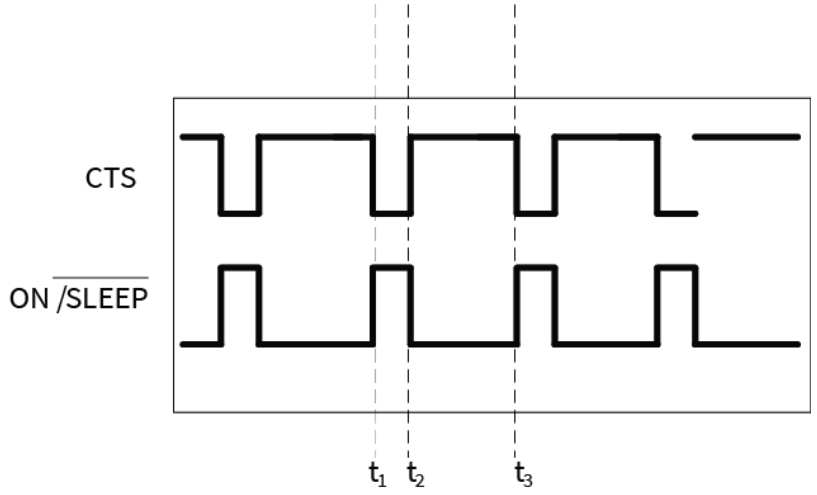


Micro cyclic sleep pins



Cyclic sleep pin waveform

The following figure shows the cyclic sleep waveforms.



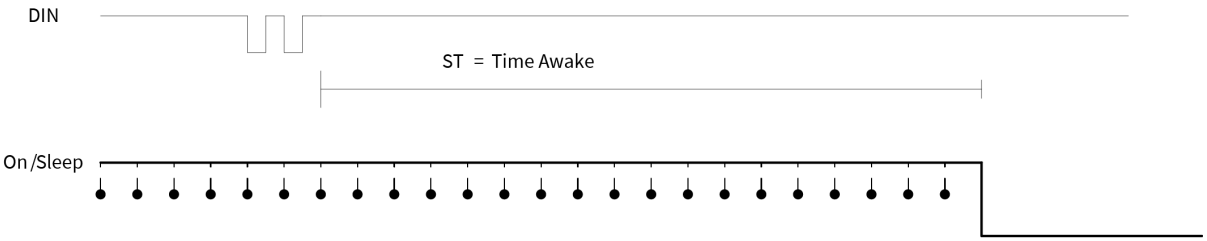
In the figure above, t1, t2, and t3 represent the following events:

- t1 - Time when the device wakes from cyclic sleep
- t2 - Time when the device returns to sleep
- t3 - Later time when the device wakes from cyclic sleep

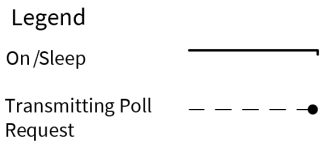
The wake time and sleep time are configurable with software commands.

Wake time (until sleep)

In cyclic sleep mode (**SM** = **4** or **5**), if the device receives serial or RF data, it starts a sleep timer (time until sleep). Any data received serially or over the RF link restarts the timer. Set the sleep timer value with ST (Cyclic Sleep Wake Time). While the device is awake, it sends poll request transmissions every 100 ms to check its parent for buffered data messages. The device returns to sleep when the sleep timer expires, or if it receives SI (Sleep Immediately) as shown in the following image.



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



Sleep period

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

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

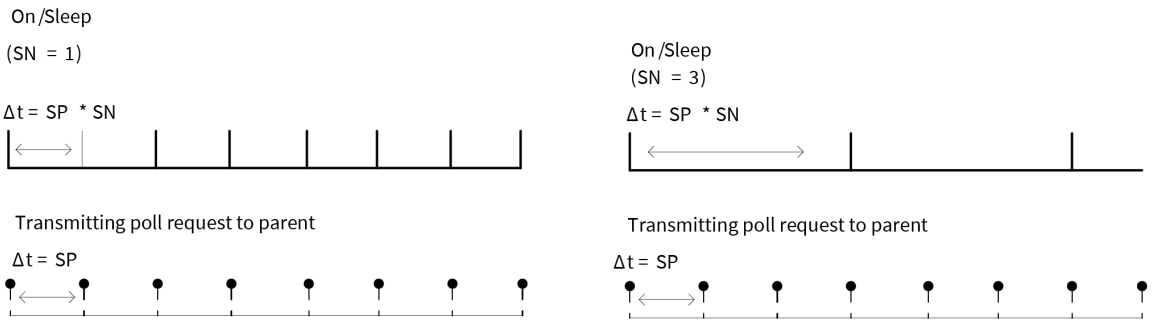
The device supports both a Short cyclic sleep and an Extended cyclic sleep that make use of these commands. These two modes allow the sleep period to be configured according to the application requirements.

Short cyclic sleep

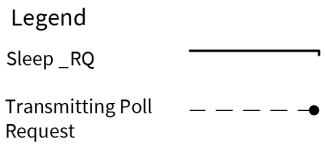
In short cyclic sleep mode, define the sleep behavior of the device by the **SP** and **SN** commands, and the **SO** command must be set to 0x00 (default) or 0x02. In short cyclic sleep mode, the **SP** command defines the sleep period and you can set it for up to 28 seconds. When the device enters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

After the sleep period expires, the XBee 3 Zigbee RF Module sends a poll request transmission to its parent to determine if the parent has any buffered data waiting for the end device. Since router and coordinator devices can buffer data for end device children up to 30 seconds, the **SP** range (up to 28 seconds) allows the end device to poll regularly enough to receive buffered data. If the parent has data for the end device, the end device starts its sleep timer (**ST**) and continues polling every 100 ms to receive data. If the end device wakes and finds that its parent has no data for it, the end device can return to sleep immediately.

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



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



Note **SP** controls the packet buffer time on routers and coordinators. Set **SP** on all router and coordinator devices to match the longest end device **SP** time. For more information, see Router/Coordinator configuration.

Extended cyclic sleep

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

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

Similar to short cyclic sleep, end devices running in this mode return to sleep when the sleep timer expires, or when they receive the **SI** command.

Deep sleep

The following are preconditions for maintaining low current draw during sleep:

- You must maintain the supply voltage within a valid operating range (2.1 to 3.6 V for the XBee, 3.0 to 3.6 V for the XBee-PRO (S2), 2.7 to 3. V for the XBee-PRO S2B).
- Each GPIO input line with a pullup resistor which is driven low draws about 100 uA current through the internal pullup resistor.
- If circuitry external to the XBee drives such input lines low, then the current draw rises above expected deep sleep levels.

- Each GPIO input line that has no pullup or pull-down resistor (is floating) has an indeterminate voltage which can change over time and temperature in an indeterminate manner.

MicroPython (with optional pin wake)

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

Recommended sleep current measurements

Properly measuring the sleep current helps to accurately estimate battery life requirements. To ensure that you take proper measurements without upsetting the normal operation of the unit under test, read the following steps.

When you measure sleep currents, it can cause problems with the devices because the equipment that measures very low currents accurately requires a large resistor in series with the power supply. This large resistor starves current from the device during a momentary wake cycle, forcing the voltage to drop to brownout levels rapidly. This voltage drop places the device in a state that may require a reset to resolve the problem.

Achieve the lowest sleep current

To achieve the lowest sleep current, you must disable brownout detectors during sleep modes. Even if the measurement equipment automatically changes current ranges, it is often too slow and cannot keep up with the necessary sudden short bursts. During long cyclic sleep periods, the device can wake every 10 to 30 seconds to reset timers and perform other necessary steps. These wake times are small and you may not notice them when measuring sleep currents.

Compensate for switching time

To compensate for the switching time of the equipment you must temporarily add an additional large cap when you need measurements to allow for short pulses of current draw (see the following schematic for details). A cap of 100 uF is enough to handle 1.5 milliseconds with 20 mA of current. You can increase or decrease the capacitor based on the switching time of the measurement circuitry and the momentary on time of the unit. Measure the leakage current of the additional cap to verify that it does not skew the low current reading. The capacitor averages the spike in current draw. The actual magnitude of the current spike is no longer visible, but you can account for the total energy consumed by integrating the current over time and multiplying by the voltage.

Internal pin pull-ups

Internal pull-up/down resistors only apply to GPIO lines that are configured as disabled (0) or digital input (3). Use PR (Pull-up/Down Resistor Enable) to enable them on a per-pin basis and use PD (Pull Up/Down Direction) to determine the direction.

Internal pin pull-ups can pull excess current and cause the sleep current readings to be higher than desired if you drive or float the pull-ups.

- Disable all pull-ups for input lines that have a low driven state during sleep.
- Enable pull-ups for floating lines or inputs that do not connect to other circuitry.

If you use an analog-to-digital converter (ADC) to read the analog voltage of a pin, it may not be possible to stop all leakage current unless you can disconnect the voltage during sleep. Each floating

input that is not at a valid high or low level can cause leakage depending on the temperature and charge buildup that you may not observe at room temperature.

Transmit RF data

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

Receiving RF data

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

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

I/O sampling

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

Wake end devices with the Commissioning Pushbutton

If you use D0 (DIO0/AD0/Commissioning Button Configuration) to enable the Commissioning Pushbutton functionality, a high-to-low transition on the AD0/DIO0 pin (Micro pin 31/SMT pin 33/TH pin 20) causes an end device to wake for 30 seconds. For more information, see Commissioning pushbutton and associate LED.

Parent verification

Since an end device relies on its parent to maintain connectivity with other devices in the network, XBee end devices include provisions to verify the connection with its parent. End devices monitor the link with their parent when sending poll messages and after a power cycle or reset event as described below.

When an end device wakes from sleep, it sends a poll request to its parent. In cyclic sleep, if the end device does not receive RF or serial data and the sleep timer is not started, it polls one time and returns to sleep for another sleep period. Otherwise, the end device continues polling every 100ms. If the parent does not send an acknowledgment response to three consecutive poll request transmissions, the end device assumes the parent is out of range, and attempts to find a new parent.

After a power-up or reset event, the end device does an orphan scan to locate its parent. If the parent does not send a response to the orphan scan, the end device attempts to find a new parent.

Rejoining

Once all devices have joined a Zigbee network, disable the permit-joining attribute disabled such that new devices are no longer allowed to join the network. You can enable permit-joining later as needed for short times. This provides some protection in preventing other devices from joining a live network.

If an end device cannot communicate with its parent, the end device must be able to join a new parent to maintain network connectivity. However, if permit-joining is disabled in the network, the end device will not find a device that is allowing new joins.

To overcome this problem, Zigbee supports rejoining, where an end device can obtain a new parent in the same network even if joining is not enabled. When an end device joins using rejoining, it performs a PAN ID scan to discover nearby networks. If a network is discovered that has the same 64-bit PAN ID as the end device, it joins the network by sending a rejoin request to one of the discovered devices. The device that receives the rejoin request sends a rejoin response if it can allow the device to join the network (that is, the child table is not full). You can use the rejoin mechanism to allow a device to join the same network even if permit-joining is disabled.

To enable rejoining, set **NJ** to less than 0xFF on the device joining. If **NJ** < 0xFF, the device assumes the network is not allowing joining and first tries to join a network using rejoining. If multiple rejoining attempts fail, or if **NJ** = **0xFF**, the device attempts to join using association.

Router/Coordinator configuration

XBee routers and coordinators may require some configuration to ensure the following are set correctly.

- RF Packet buffering timeout
- Child poll timeout
- Transmission timeout

The value of these timeouts depends on the sleep time used by the end devices.

RF packet buffering timeout

When a router or coordinator receives an RF data packet intended for one of its end device children, it buffers the packet until the end device wakes and polls for the data, or until a packet buffering timeout occurs. Use the **SP** command to set the timeout . The actual timeout is (1.2 * **SP**), with a minimum timeout of 1.2 seconds and a maximum of 30 seconds. Since the packet buffering timeout is set slightly larger than the **SP** setting, set **SP** the same on routers and coordinators as it is on cyclic sleep end devices. For pin sleep devices, set **SP** as long as the pin sleep device can sleep, up to 30 seconds.

Note In pin sleep and extended cyclic sleep, end devices can sleep longer than 30 seconds. If end devices sleep longer than 30 seconds, parent and non-parent devices must know when the end device is awake in order to reliably send data. For applications that require sleeping longer than 30 seconds, end devices should transmit an **I/O** sample or other data when they wake to alert other devices that they can send data to the end device.

Child poll timeout

Router and coordinator devices maintain a timestamp for each end device child indicating when the end device sent its last poll request to check for buffered data packets. If an end device does not send a poll request to its parent for a certain period of time, the parent will discard the packet.

Set the child poll timeout with the **SP** and **SN** commands. **SP** and **SN** should be set such that **SP** * **SN** matches the longest expected sleep time of any end devices in the network. The device calculates the actual timeout as (3* **SP** * **SN**), with a minimum of five seconds. For networks consisting of pin sleep end devices, set the **SP** and **SN** values on the coordinator and routers so the **SP** * **SN** matches the longest expected sleep period of any pin sleep device.

Adaptive polling

PO (Polling Rate) determines the regular polling rate. However, if RF data has been recently received by an end device, it is likely that more RF data could be on the way. Therefore, the end device polls at a faster rate, gradually decreasing its adaptive poll rate until polling resumes at the regular rate as defined by the **PO** command.

Transmission timeout

When you are sending RF data to a remote router, because routers are always on, the timeout is based on the number of hops the transmission may traverse. Set the timeout using NH (Maximum Unicast Hops). For more information, see Transmission, addressing, and routing.

Since end devices may sleep for lengthy periods of time, the transmission timeout to end devices also allows for the sleep period of the end device. When sending data to a remote end device, the transmission timeout is calculated using the **SP** and **NH** commands. If the timeout occurs with no acknowledgment received, the source device re-sends the transmission until it receives an acknowledgment, up to two more times.

The transmission timeout per attempt is:

3 * ((unicast router timeout) + (end device sleep time))

3 * ((50 * **NH**) + (1.2 * **SP**)), where **SP** is measured in 10 ms units.

Short sleep periods

Pin and cyclic sleep devices that sleep less than 30 seconds can receive data transmissions at any time since their parent devices are able to buffer data long enough for the end devices to wake and poll to receive the data. Set **SP** the same on all devices in the network. If end devices in a network have more than one **SP** setting, set **SP** on the routers and coordinators to match the largest **SP** setting of any end device. This ensure the RF packet buffering, poll timeout, and transmission timeouts are set correctly.

Extended sleep periods

Pin and cyclic sleep devices that might sleep longer than 30 seconds cannot receive data transmissions reliably unless you take certain design approaches. Specifically, the end devices should use I/O sampling or another mechanism to transmit data when they wake to inform the network they can receive data. **SP** and **SN** should be set on routers and coordinators such that (**SP** * **SN**) matches the longest expected sleep time.

As a general rule, **SP** and **SN** should be set the same on all devices in almost all cases.

Sleep examples

Some sample XBee configurations to support different sleep modes follow. In Command mode, issue each command with a leading **AT** and no **=** sign, for example, **ATSM4**. In the API, the two byte command is used in the command field, and parameters are populated as binary values in the parameter field.

Example 1: Configure a device to sleep for 20 seconds, but set SN such that the On/sleep line will remain de-asserted for up to 1 minute.

The following settings should be configured on the end device.

- **SM** = 4 (cyclic sleep) or 5 (cyclic sleep, pin wake).
- **SP** = 0x7D0 (2000 decimal). This causes the end device to sleep for 20 seconds since **SP** is measured in units of 10 ms.
- **SN** = 3. (With this setting, the On/Sleep pin asserts once every 3 sleep cycles, or when it receives RF data) **SO** = 0.

Set all router and coordinator devices on the network **SP** to match **SP** on the end device. This set the RF packet buffering times and transmission timeouts correctly.

Since the end device wakes after each sleep period (**SP**), you can set the **SN** command to 1 on all routers and the coordinator.

Example 2: Configure an end device to sleep for 20 seconds, send 4 I/O samples in 2 seconds, and return to sleep.

Because **SP** is measured in 10 ms units, and **ST** and **IR** are measured in 1 ms units, configure an end device with the following settings:

- **SM** = 4 (cyclic sleep) or 5 (cyclic sleep, pin wake).
- **SP** = 0x7D0 (2000 decimal). This causes the end device to sleep for 20 seconds.
- **SN** = 1.
- **SO** = 0.
- **ST** = 0x7D0 (2000 decimal). This sets the sleep timer to 2 seconds.
- **IR** = 0x258 (600 decimal). Set **IR** to a value greater than (2 seconds / 4) to get 4 samples in 2 seconds. The end device sends an I/O sample at the **IR** rate until the sleep timer has expired.

You must enable at least one analog or digital I/O line for I/O sampling to work. To enable AD1/DIO1 (Micro pin 30/SMT pin 32/TH pin 19) as a digital input line, you must set the following:

D1 = 3

Set all router and coordinator devices on the network **SP** to match **SP** on the end device. This ensures that RF packet buffering times and transmission timeouts are set correctly.

Example 3: configure a device for extended sleep: to sleep for 4 minutes.

- **SP** and **SN** must be set such that **SP** * **SN** = 4 minutes. Since **SP** is measured in 10 ms units, use the following settings to obtain 4 minute sleep.
- **SM** = 4 (cyclic sleep) or 5 (cyclic sleep, pin wake) **SP** = 0x7D0 (2000 decimal, or 20 seconds).
- **SN** = 0x0B (12 decimal).
- **SO** = 0x04 (enable extended sleep).

With these settings, the module sleeps for **SP** * **SN** time, or (20 seconds * 12) = 240 seconds = 4 minutes.

For best results, the end device should send a transmission when it wakes to inform the coordinator (or network) when it wakes. It should also remain awake for a short time to allow devices to send data to it. The following are recommended settings.

- **ST** = 0x7D0 (2 second wake time)
- **SO** = 0x06 (enable extended sleep and wake for ST time)
- **IR** = 0x800 (send 1 I/O sample after waking). Enable at least one analog or digital I/O sample enabled for I/O sampling.

With these settings, the end device wakes after 4 minutes and sends 1 I/O sample. It then remains awake for 2 seconds before returning to sleep.

Set **SP** and **SN** to the same values on all routers and coordinators that could potentially allow the end device to join. This ensures the parent does not timeout the end device from its child table too quickly.

The **SI** command can optionally be sent to the end device to cause it to sleep before the sleep timer expires.

I/O support

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

Digital I/O support

Digital I/O is available on lines DIO0 through DIO12 (D0 (DIO0/AD0/Commissioning Button Configuration) - D9 (DIO9/ON_SLEEP) and P0 (DIO10/RSSI Configuration) - P4 (DIO14/DIN Configuration)). 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	Micro Pin	SMT Pin	TH Pin	AT Command
DIO0	31	33	20	D0 (DIO0/AD0/Commissioning Button Configuration)
DIO1	30	32	19	D1 (AD1/DIO1/TH_SPI_ATTN Configuration)
DIO2	29	31	18	D2 (DIO2/AD2/TH_SPI_CLK Configuration)
DIO3	28	30	17	D3 (DIO3/AD3/TH_SPI_SSEL Configuration)
DIO4	23	24	11	D4 (DIO4/TH_SPI_MOSI Configuration)
DIO5	26	28	15	D5 (DIO5/Associate Configuration)
DIO6	27	29	16	D6 (DIO6/RTS)
DIO7	24	25	12	D7 (DIO7/CTS)
DIO8	9	10	9	D8 (DIO8/DTR/SLP_RQ)
DIO9	25	26	13	D9 (DIO9/ON_SLEEP)
DIO10	7	7	6	P0 (DIO10/RSSI Configuration)
DIO11	8	8	7	P1 (DIO11 Configuration)
DIO12	5	5	4	P2 (DIO12/TH_SPI_MISO Configuration)
DIO13	3	3	2	P3 (DIO13/DOUT Configuration)
DIO14	4	4	3	P4 (DIO14/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/AD0/Commissioning Button Configuration)
ADC1	30	32	19	D1 (AD1/DIO1/TH_SPI_ATTN Configuration)
ADC2	29	31	18	D2 (DIO2/AD2/TH_SPI_CLK Configuration)
ADC3	28	30	17	D3 (DIO3/AD3/TH_SPI_SSEL Configuration)
PWM0	7	7	6	P0 (DIO10/RSSI Configuration)
PWM1	8	8	7	P1 (DIO11 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 (I/O 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 (Digital Change Detection))
  - 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 (Force 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

Regardless of how I/O data is generated, the format of the sample data is always represented as a series of bytes in the following format:

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

Bytes	Name	Description
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.

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

On-demand sampling

You can use IS (Force 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:

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)
03	Analog channel mask, indicates which analog lines are sampled (0x03 = 0000 00**11**b = AD0, 1)

Output	Description
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

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)
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

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)
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 (I/O 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

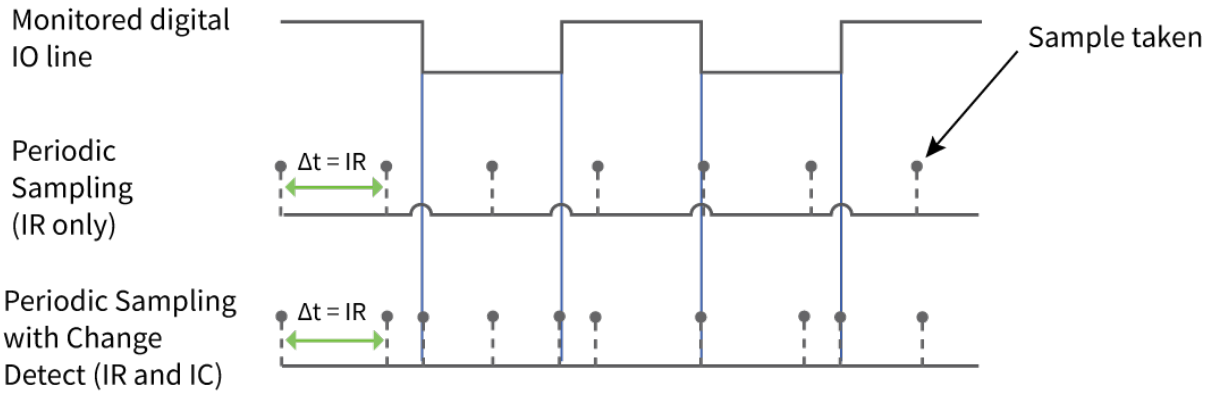
If the receiving device is operating in API operating mode the I/O sample data format is emitted out of the serial port. Devices that are in Transparent operating mode discard the I/O data samples they receive unless you enable line passing.

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 (Digital Change Detection) 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.

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.

AT commands

Networking commands

This section lists the AT commands that affect the operation of the Zigbee network and joining device behavior.

CE (Device Role)

Determines whether the device should form or join a network.

When forming a network, the device acts as a Zigbee network manager/coordinator. Sleep must be disabled before **CE** can be set.

Changing **CE** after network association causes the device to leave the network.

Parameter range

0 - 1

Parameter	Description
0	Join Network
1	Form Network (**SM** must be 0 to set **CE** to 1)

Default

0

ID (Extended PAN ID)

The preconfigured Extended PAN ID used when forming or joining a network.

ID restricts joining to only networks with a matching Operating Pan (**OP**) value. If **ID** is set to 0, the device attempt to join any open network.

When forming a network (**CE** = **1**), **ID** preconfigures the Extended PAN ID used to form the network. When you set **ID** to 0, a random Extended PAN ID is generated.

Changing **ID** after network association causes the device to leave the network.

Parameter range

0 - 0xFFFFFFFFFFFFFFFF

Default

0

II (Initial 16-bit PAN ID)

The preconfigured 16-bit PAN ID used when forming a network. Use this command to replace a coordinator node on an existing unencrypted Zigbee network.

When you set **II** to the default value (recommended) the device forms a network on a random 16-bit PAN ID.

Changing **II** on the coordinator after the network is formed causes it to leave and form a new network.

Range

0 - 0xFFFF

Default

0xFFFF

ZS (Zigbee Stack Profile)

Set or read the initial Zigbee stack profile used by the device. This parameter must be the same on all devices joining the same network. If XBee devices are the only type of radio on your network, leave **ZS** at the default value of 0; a non-zero value allows third-party Zigbee devices to join.

If operating in Command mode, any changes to **ZS** is made active only when Command mode exits (via timeout or CN (Exit Command mode)). Changing **ZS** causes all current parameters to be written to persistent storage and the module restarts; this is equivalent to issuing **WR** and **FR** commands.

When the device restarts as a result of changing **ZS** or **C8**, no modem status is generated. $\overline{\text{CTS}}$ will also de-assert during this period, so flow control is advised. If hardware flow control is not being used, a 1-second delay after exiting Command mode (or applying changes if using API) may be necessary to avoid data loss.

Changing **ZS** after network association causes the device to leave the network.

Parameter range

0 - 2

Parameter	Description
0	Digi Proprietary
1	Zigbee 2006 (legacy)
2	Zigbee-PRO (third-party)

Default

0

CR (Conflict Report)

The number of PAN ID conflict reports that must be received by the network manager within one minute to trigger a PAN ID change.

A corrupt beacon can cause a report of a false PAN ID conflict.

A higher value reduces the chance of a false PAN ID change.

A value of zero disables automatically changing the PAN ID due to PAN ID conflicts. In this case, if a PAN ID conflict is detected and API mode is enabled (**AP** = **1** or **2**), the coordinator will emit a modem status value of 0x3E.

Parameter range

0 - 0x3F

Default

3

NJ (Node Join Time)

Configure the amount of time the local device's join window is open for. The join window specified by **NJ** only affects the window for the local node and does not affect the timing of the rest of the network. This value can be changed at run time without requiring a Coordinator or Router to restart.

Zigbee 3.0 does not allow the network to be always open for joining; modules that attempt to join when the join window is closed will report an **AI** value of 0x23. The join window can optionally be persistently opened by setting **NJ** = **0xFF**, but this causes the device to operate outside of the Zigbee 3.0 specifications.

See Join window for information on the join window and what circumstances can cause it to open.

When the Join Window is opened, the Association LED will blink rapidly to indicate that joining is allowed. If operating in API mode (**AP** = **1** or **2**), a 0x8A Modem Status frame will be generated when the state of the join window changes:

- 0x43 - Join window is open
- 0x44 - Join window is closed

If you set **NJ** to 0, the join window will always shut and be closed; this is the recommended setting for secure networks. When configured with this setting, using a **CB2** AT command or pressing the commissioning button twice opens the join window for one minute.

On end devices, **NJ** also enables or disables rejoining attempts. For an end device to enable rejoining, set **NJ** less than **0xFF** on the device that joins. If **NJ** < **0xFF**, the device assumes the network is not allowing joining and first tries to join a network using rejoining. If multiple rejoining attempts fail, or if **NJ** = **0xFF**, the device attempts to join using association.

Note When a device is rejoining a network, the join window does not need to be open. However, if the rejoin attempt fails six times, the module attempts to join by association which requires an open joining window.

Parameter range

0 - 0xFF (seconds)

Default

0xFE (254 seconds)

DJ (Disable Joining)

Prevent a local device from joining a network.

This does parameter does not affect end devices that are already joined to a network. It only prevents those devices from joining another network.

Note This parameter is not written to flash with the **WR** command and reverts to default after a power cycle.

Parameter range

0 - 1

Parameter	Description
0	Enable Joining
1	Disable Joining

Default

1

NR (Network Reset)

Resets network layer parameters on one or more modules within a PAN. Responds immediately with an **OK** then causes a network restart. The device loses all network configuration and routing information.

If **NR** = 0: Resets network layer parameters on the node issuing the command.

If **NR** = 1: Sends broadcast transmission to reset network layer parameters on all nodes in the PAN.

Note **NR** and **NR0** both perform the same function and may be used interchangeably.

Parameter range

0 - 1

Default

N/A

NW (Network Watchdog Timeout)

Set the network watchdog timeout used to ensure that a coordinator is active on the network (for example, a keep alive message).

If **NW** is set > 0, the router monitors communication from the coordinator (or data collector) and leaves the network if it cannot communicate with the coordinator for 3 **NW** periods. Alternatively, If **DC** bit 5 is set, the router will not leave the network but will instead attempt to rejoin the coordinator. The device resets the timer each time it receives or sends data to a coordinator, or if it receives a many-to-one broadcast.

Parameter range

0 - 0x64FF [x 1 minute](up to approximately 18 days)

Default

0 (disabled)

JV (Coordinator Join Verification)

Used during join and rejoin attempts to determine if a coordinator is present on the target network. This verification option is only applicable for a Distributed Trust Center (**EO** = **0**) or unencrypted network (**EE** = **0**). On a Centralized Trust Center network (**EO** = **2**), the coordinator is required to be present for devices to associate so **JV** will have no effect.

If **JV** = **1**, a router or end device verifies the coordinator is on its operating channel when joining or coming up from a power cycle. If a coordinator is not detected, the router or end device leaves its

current channel and attempts to join a new PAN. If **JV** = **0**, the router or end device continues operating on its current channel even if a coordinator is not detected.

Parameter range

0 - 1

Parameter	Description
0	No coordinator verification
1	Coordinator verification enabled

Default

0

JN (Join Notification)

Broadcast Join Notification upon successful join attempt.

If enabled, the device transmits a broadcast node identification packet on power up and when joining. This action blinks the Associate LED rapidly on all devices that receive the transmission, and sends an API frame out the serial port of API devices.

Digi recommends you disable this feature for large networks to prevent excessive broadcasts.

Parameter range

0 - 1

Parameter	Description
0	Disabled
1	Broadcast notification to network upon joining

Default

0

DO (Miscellaneous Device Options)

A bitfield that contains advanced device options that do not have dedicated AT commands.

Leave unused bits clear so future device options are not inadvertently enabled during a firmware update.

Bit field:

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

Bit	Description
0	Reserved.
1	Reserved.

Bit	Description
2	Reserved.
3	Reserved.
4	Disable Tx packet extended timeout.
5	Disable ACK for end device I/O sampling.
6	Enable High-RAM concentrator. Set this bit to **0** for Low-RAM concentrator, where Route Record Indicator - 0xA1 frames will be emitted for external storage and use in subsequent Create Source Route - 0x21 frames. This option will only take effect when **AR** < **0xFF** or when acting as a Centralized Trust Center.
7	When the Network Watchdog triggers, search for a coordinator on a new network to join. The Network Watchdog must be enabled for this to take effect—NW (Network Watchdog Timeout) > **0**. With this bit set, the device will remain on the current network until a valid coordinator is found. If the coordinator is found on a different network, the device will leave the current network and join the new network. See Network Locator option.

Parameter range

0 - 0xFF

Default

0x40

DC (Joining Device Controls)

A bitfield that contains advanced joining device controls that do not have dedicated AT commands. These options only apply to joining devices (**CE**=0).

Leave unused bits clear so future device controls are not inadvertently enabled during a firmware update.

Bit field:

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

Bit	Description
0	Generate a preconfigured link key using device's install code (**KY** is ignored). Enabling this option requires the joining device be registered to the trust center.
1	Ignore NWK leave requests after joining.
2	Enable verbose join information.
3	Join network with best response (strongest signal) instead of first responder.
4	Reserved
5	An orphaned router will not leave the network but will attempt to rejoin the coordinator indefinitely. This functionality also requires the network watchdog to be enabled (**NW** > **0**).

Parameter range

0 - 0xFFFF

Default

0

C8 (Compatibility Options)

A bitfield that contains options for compatibility with legacy XBee Zigbee devices.

Devices prior to the XBee 3 use a different scale to represent LQI. **C8** bit 4 (**C8** | 0x10) enables an LQI compatibility mode. Networks that contain a mix of XBee 3 Zigbee and legacy XBee devices should enable this feature. Otherwise operating a mixed network without this bit set will prioritize legacy devices when determining route cost.

If operating in Command mode, changing **C8** bit 4 is made active only when Command mode exits via timeout or **CN** (Exit Command mode). Changing this bit causes all current parameters to be written to persistent storage and the device restarts; this is equivalent to issuing **WR** and **FR** commands. When the device restarts, no modem status is generated. CTS will also de-assert during this period, so flow control is advised. If hardware flow control is not being used, a 1-second delay after exiting Command mode (or applying changes if using API) may be necessary to avoid data loss.

Changing **C8** after network association causes the device to leave the network.

Parameter range

0x00, 0x10

Bit field:

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

Bit	Meaning
0	Reserved
1	Reserved
2	Reserved
3	Reserved
4	Legacy LQI Calculation Compatibility

Default

0x00

Discovery commands

Network Discovery and corresponding discovery options.

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 Discovery) 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 0 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

0x120000

NT (Node Discover Timeout)

Sets the amount of time a base node waits for responses from other nodes when using the ND (Network Discovery) 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

0x20 - 0xFF (x 100 ms)

Default

0x3C (6 seconds)

NO (Network Discovery Options)

Set the Advanced Options that affect how a particular device responds to network discoveries (**ND** and **DN** commands) and when sending a node identification.

Bit field:

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

Bit	Meaning
0	Append the **DD** (Digi Device Identifier) value to **ND** responses and node identification frames.
1	Local device sends its own **ND** response when **ND** is issued.

Parameter range

0 - 3

Default

0x0

ND (Network Discovery)

Discovers and reports all of the devices it finds on a network. The command reports the following information after a jittered time delay (based on the local device's **NT** value).

MY<CR> (2 bytes) (always 0xFFFE)

SH<CR> (4 bytes)

SL<CR> (4 bytes)

DB<CR> (Contains the detected signal strength of the response in negative dBm units)

NI <CR> (variable, 0-20 bytes plus 0x00 character)

PARENT_NETWORK ADDRESS<CR> (2 bytes)

DEVICE_TYPE<CR> (1 byte: **0** = Coordinator, **1** = Router, **2** = End Device)

STATUS<CR> (1 byte: reserved)

PROFILE_ID<CR> (2 bytes)

MANUFACTURER_ID<CR> (2 bytes)

DIGI DEVICE TYPE<CR> (4 bytes. Optionally included based on **NO** settings.)

RSSI OF LAST HOP<CR> (1 byte. Optionally included based on **NO** settings.)

After (**NT** * 100) milliseconds, the command ends by returning a <CR>.

If you send **ND** through a local API frame, each network node returns a separate Local or Remote AT Command Response API packet, respectively. The data consists of the previously listed bytes without the carriage return delimiters. The **NI** string ends in a "0x00" null character because it is a variable length.

ND also accepts a **NI** (Node Identifier) as a parameter (optional). In this case, only a device that matches the supplied identifier responds after a jittered time delay. If there are no matching devices, the command returns an "ERROR".

The radius of the **ND** command is set by the **BH** command.

A status code of 1=ERROR will be returned if the transmit queue is full. That means there are already four messages queued for transmission. The application is trying to send messages faster than the device can process the requests. The application may either try again later, be redesigned to send messages at a slower rate, or wait for a Tx Status response for a prior message before attempting to send another.

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

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 Local AT Command Request - 0x08:

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.

Parameter range

Up to 20-byte printable ASCII string

Default

N/A

AS (Active Scan)

Forces an active scan of the neighborhood for beacon responses. The **AS** command cannot be issued remotely.

An Active scan returns a multi-line response with each field separated by a carriage return:

AS_type – unsigned byte = Always returns 2, indicating the protocol is Zigbee

Channel – unsigned byte

PAN – unsigned word in big endian format

Extended PAN – eight unsigned bytes in bit endian format

Allow Join – unsigned byte – 1 indicates join is enabled, 0 that it is disabled

Stack Profile – unsigned byte

LQI – Link Quality Indicator - unsigned byte, higher values are better

RSSI – Relative Signal Strength Indicator - signed byte, lower values are better

Each field in the AS response is separated by a carriage return (0x0D character).

An additional carriage return separates multiple beacons.

Two additional carriage returns 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

Operating Network commands

The following read-only AT commands provide information about the attached Zigbee network.

AI (Association Indication)

Read information regarding last node join request. Query **AI** during a join attempt to identify the current state.

You can also enable Verbose Joining (**DC**=4) to debug a join attempt in real-time.

Status code	Meaning
0x00	Successfully formed or joined a Zigbee network.
0x21	Scan found no PANs.
0x22	Scan found no valid PANs based on SC and ID settings.
0x23	Valid PAN found, but joining is currently disabled.
0x24	No joinable beacons were found.
0x27	Join attempt failed.
0x2A	Failed to start coordinator.
0x2B	Checking for existing coordinator.
0x40	Secure Join - Successfully attached to network, waiting for new link key.
0x41	Secure Join - Successfully received new link key from the trust center.
0x44	Secure Join - Failed to receive new link key from the trust center.
0xAB	Attempted to join a device that did not respond.
0xAD	Secure Join - a network security key was not received from the trust center.
0xAF	Secure Join - a preconfigured key is required to join the network.
0xFF	Initialization time; no association status has been determined yet.

Parameter range

0 - 0xFF [read-only]

Default

N/A

OP (Operating Extended PAN ID)

Read the 64-bit extended PAN ID of the attached network. The **OP** value reflects the operating 64-bit extended PAN ID where the device is running.

Parameter range

1 - 0xFFFFFFFFFFFFFFFF

Default

N/A

OI (Operating 16-bit PAN ID)

Read the 16-bit PAN ID of the attached network. The **OI** value reflects the actual 16-bit PAN ID where the device is running.

Parameter range

0 - 0xFFFF [read-only}

Default

N/A

CH (Operating Channel)

Read the channel number of the attached network. Channels are represented as IEEE 802.15.4 channel numbers.

A value of 0 means the device has not joined a PAN and is not operating on any channel.

Parameter range

0, 0x0B - 0x1A (Channels 11 through 26) [read-only]

Default

N/A

NC (Number of Remaining Children)

Read the number of remaining end device children that can join the device. If **NC** returns 0, the device is at capacity and cannot allow any more end device children to join.

Parameter range

0 - 0x14 (20 child devices)

Default

N/A

Zigbee Addressing commands

The following AT commands are used for communication with a Zigbee network after association.

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 Network Address)

Reads the 16-bit network address of the device, which is randomly assigned by the network manager upon association.

A value of 0xFFFE means the device has not joined a Zigbee network.

Parameter range

0 - 0xFFFF [read-only]

Default

0 - 0xFFFE

MP (16-bit Parent Network Address)

Read the 16-bit network address of the end device's parent. A value of **0xFFFE** means the device does not have a parent or is not configured as an end device.

Parameter range

0 - 0xFFFE [read-only]

Default

0xFFFE

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.

Reserved Zigbee network addresses:

- **0x000000000000FFFF** is a broadcast address (**DH** = **0**, **DL** = **0xFFFF**).
- **0x0000000000000000** addresses the network coordinator.

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 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.

Reserved Zigbee network addresses:

- **0x000000000000FFFF** is a broadcast address (**DH** = **0**, **DL** = **0xFFFF**).
- **0x0000000000000000** addresses the network coordinator.

Parameter range

0 - 0xFFFFFFFF

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) and route repair for unicast traffic.

Bit	Meaning
4	Send data securely—requires secure session be established with destination. Enabling this bit will reduce maximum payload size by 4 bytes.
5	Enable APS end-to-end encryption (if **EE** = **1**). Enabling this bit will reduce the maximum payload size by 9 bytes.
6	Use extended timeout.

Default

0

NP (Maximum Packet Payload Bytes)

If operating in Transparent mode (**AP** = **0**), **NP** reads the maximum number of RF payload bytes that you can typically send in a transmission based on current parameter settings (**DH**, **DL**, **TO**, and **EE**). Transmissions in Transparent mode do not use fragmentation and this value represents the payload of a single fragment. For other operating modes, **NP** returns the typical maximum number of RF payload bytes that can be transmitted with fragmentation enabled (255 bytes).

See Maximum RF payload size for more information.

Some options may impact maximum payload size that are not captured by the **NP** value: sending a packet securely across a Secure Session (API transmit option bit 4 enabled) will reduce the maximum payload size by 4 bytes. Using source routing (**AR** < **0xFF**) further reduces the maximum payload size depending on how many hops are traversed.

Using source routing (**AR** < 0xFF), further reduces the maximum payload size depending on how many hops are traversed.

Note **NP** returns a hexadecimal value. For example, if **NP** returns 0x54, this is equivalent to 84 bytes.

Parameter range

0 - 0xFF [read-only]

Default

N/A

Zigbee configuration commands

The following AT commands adjust the advanced communication settings that affect outgoing data transmissions in a Zigbee network.

NH (Maximum Unicast Hops)

This parameter determines the timeout value used for unicast transmissions from the local device.

The timeout is computed as (50 * **NH**) + 100 ms. A unicast transmission that does not receive an acknowledgement within the timeout period is reported as a failed transmission.

The default unicast timeout of 1.6 seconds (**NH**=0x1E) is enough time for data and the acknowledgment to traverse approximately 8 hops.

If BH (Broadcast Hops) = **0**, **NH** is used to set the maximum number of hops across the network when sending a broadcast transmission. **NH** is also used to set the maximum number of hops for broadcast if **BH** > **NH**.

Parameter range

0 - 0xFF

Default

0x1E

BH (Broadcast Hops)

The number of hops that broadcast transmissions from the local device traverse. Unlike **NH**, this parameter is a fixed number of hops and not used in timeout calculations.

Parameter range

0 - 0x1E

Default

0

AR (Aggregate Routing Notification)

Set or read the periodic time for broadcasting aggregate route messages. Setting **AR** enables many-to-one routing from the broadcasting device using the concentrator mode determined by **DO** Bit 6.

Set **AR** to 0x00 to send only one broadcast.

Set **AR** to 0xFF to stop sending broadcasts (many-to-one routing will still be enabled until a network reset occurs).

Parameter range

0 - 0xFF (x10 sec)

Default

0xFF (disabled)

SE (Source Endpoint)

Sets or displays the application layer source endpoint value used for data transmissions.

This command only affects outgoing transmissions in Transparent mode (**AP** = **0**).

Note Endpoints **0xDC** - **0xEE** are reserved for special use by Digi and should not be used in an application outside of the listed purpose.

The reserved Digi endpoints are:

- 0xE8 - Digi data endpoint
- 0xE6 - Digi device object endpoint
- 0xE5 - Secure Session Server endpoint

- 0xE4 - Secure Session Client endpoint
- 0xE3 - Secure Session SRP authentication endpoint

Parameter range

0 - 0xFF

Default

0xE8

DE (Destination Endpoint)

Sets or displays the application layer destination endpoint used for data transmissions.

This command only affects outgoing transmissions in Transparent mode (**AP** = **0**).

Note Endpoints **0xDC** - **0xEE** are reserved for special use and should not be used in an application outside of the listed purpose.

The reserved Digi endpoints are:

- 0xE8 - Digi data endpoint
- 0xE6 - Digi device object endpoint
- 0xE5 - Secure Session Server endpoint
- 0xE4 - Secure Session Client endpoint
- 0xE3 - Secure Session SRP authentication endpoint

Parameter range

0 - 0xFF

Default

0xE8

CI (Cluster ID)

The application layer cluster ID value. The device uses this value as the cluster ID for all data transmissions in Transparent mode and for all transmissions performed with the Transmit Request - 0x10 in API mode. In API mode, transmissions performed with the Explicit Addressing Command Request - 0x11 ignore this parameter.

- **0x11** is a transparent data cluster ID.
- **0x12** is a loopback cluster ID. The destination node echoes any transmitted packet back to the source device.

Parameter range

0 - 0xFFFF

Default

0x11 (Transparent data cluster ID)

Security commands

The following AT commands are used to set the initial security parameters.

Note Configure these parameters prior to forming/joining a network. Changing these parameters may cause the node to leave any currently attached network.

EE (Encryption Enable)

Set or read the encryption enable setting of the local device.

Parameter range

0 - 1

Parameter	Description
0	Encryption Disabled
1	Encryption Enabled

Default

0

EO (Encryption Options)

A bitfield that contains advanced encryption options that do not have dedicated AT commands. These options are only applicable when encryption is enabled (**EE** = **1**).

Leave unused bits clear so future encryption options are not inadvertently enabled during a firmware update.

Bit field:

Unused bits must be set to **0**.

Note When changing the **EO** option on a router or sleeping end device, you need to send an NR (Network Reset) for the new options to take effect if the device is already joined to a network.

These bits may be logically OR'ed together:

Bit	Description
0	Send/receive NWK keys in the clear (unsecure).
1	1 = Centralized Trust Center. 0 = Distributed Trust Center.
2	Use EUI64-hashed link keys (used on centralized trust center only).
3	Emit join notification frames (used on centralized trust center only).
4	Allow joining using well-known default link keys (unsecure).

Parameter range

0 - 0xFFFF

Default

2

KY (Link Key)

The preconfigured link key used during network formation and joining. When queried, **KY** returns zero if the value of the key is zero; for all other values it returns an **OK** response to indicate that a key is present.

On a forming node (**CE** = **1**):

KY acts as the preconfigured global link key of the trust center. If you set **KY** to **0**, a random link key will be generated and used to form the network; this requires joining devices to be registered to the trust center using a 0x24 registration API frame.

On a joining node (**CE** = **0**):

KY is the preconfigured link key used during joining; it must either match the **KY** value set on the trust center or be registered with the trust center via 0x24 registration frame. If you set **KY** to **0** on a joining node, an unsecure well-known default link key will be used. **EO** bit 4 must be set on the trust center for unsecure devices configured in this way to join.

Parameter range

0 - 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF [write-only]

Default

0

NK (Trust Center Network Key)

The network key used by the trust center to encrypt network traffic. If you set **NK** to 0 (recommended), a random network key is used. **NK** is not used by joining nodes, as the network key is securely obtained as part of the join process. When queried, **NK** returns zero if the value of the key is zero; for all other values it returns an **OK** response to indicate that a key is present.

If operating with a centralized trust center (**EE** = **1**, **EO** = **2**), **NK** can be changed to rotate the network key, which will be distributed to every device on the network. In a distributed trust center, every router has a copy of the network key, so it cannot be changed after the network is formed.

When the network key is changed, a Modem Status - 0x8A of **0x45** will be emitted. After a period of time, a 0x07 modem status will indicate that the network has switched to the new key.

Parameter range

0 - 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF [write-only]

Default

0

RK (Trust Center Network Key Rotation Interval)

Used by a centralized trust center to automatically rotate the network key. **RK** determines in the interval, in days, in which a new random network key is generated and distributed to the network.

Automatic network key rotation can only be performed if **NK** is set to **0** and the device is acting as a centralized trust center (**CE** = **1**, **EE** = **1**, **EO** | **2**). After **RK** days of runtime, the network key is rotated and the network is updated with the new key.

Setting **RK** to **0** performs a one-time network key rotation. This can be used by an external means to extend the key rotation beyond the maximum of 22 days or to securely rotate keys without explicitly setting **NK**.

Devices on the network store the current and previous network keys to ensure devices remain on the network through long sleep cycles or periods of lost connectivity. Should a device miss a network key update, it will securely rejoin the network and obtain the new network key from the trust center.

When a network key rotation is initiated, a Modem Status - 0x8A of **0x45** is emitted. After a period of time, a **0x07** modem status will indicate that the network has switched to the new key.

Parameter range

0 - 0x16 (days)

Default

0x16

KT (Trust Center Link Key Registration Timeout)

When registering a joining device using a 0x24 registration API frame, this parameter determines the length of time the key table entry persists before expiring.

This timeout is separate from the **NJ** join time. The join window opens when a device is successfully registered to the trust center via the 0x24 Device Registration API Frame.

Parameter range

0x1E - 0xFFFF (seconds)

Default

0x12C (500 Seconds)

I? (Install Code)

The install code is a random key assigned to every Zigbee 3.0 device at the factory. This install code can be used to securely register a device to a trust center using a 0x24 registration frame and option bit.

For the install code to be used by the joining device, **DC** bit 0 must be set on the joiner.

Parameter range

0 - 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF [read-only]

Default

Set in the factory.

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 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 - 0x1C (bit field)

Default

0

BK (Centralized Trust Center Backup and Restore)

BK is used to create a backup file in the file system named **backup_TC.xbee**. This command is also used to restore a trust center using a backup file stored in the file system.

USAGE:

BK 0

Passing a parameter of zero creates a unique backup file using the device's **SL** and stores it in the centralized trust center's file system. To create a backup file, the module must be configured as coordinator (**CE** = **1**), as a centralized trust center (**EO** = **2**), with security enabled (**EE** = **1**), and have a **KB** key set.

BK 1 <Backup Filename> <CX Output>

Note The backup filename is not an optional parameter.

Passing a parameter of 1 causes the device to look for a backup file in the file system and uses it to set the device's configuration to match the original coordinator.

Passing a parameter of 1 and **<Backup Filename>** causes the device to look for a backup file with the given name in the file system and uses it to set the device's configuration to match the original coordinator.

After restoring the backup, if the new coordinator fails to communicate, the **CX** command may need to be used to refresh needed network information in the new coordinator. The values returned by executing the **CX** command on a router that is part of the existing network can be passed as optional parameters to the new coordinator using the **BK 1 <Backup Filename>** command.

CX (Centralized Trust Center Network Information Update)

CX is a read-only, router specific command that returns three hexadecimal numbers that can be entered as optional parameters to the **BK** command during the centralized trust center restore operation. The **CX** command is only applicable for routers and will return **ERROR** if executed on a coordinator.

Example

The **CX** command when executed on a router will produce similar output to:

ATCX

FE 7D48 2

These numbers can then be used as optional parameters when issuing a **BK 1 <Backup Filename>** to a new device that is replacing an inoperable centralized trust center:

ATBK 1 "backup_TC417AD47A.xbee" FE 7D48 2

OK

KB (Centralized Trust Center Backup Key)

KB is used to set the 256-bit centralized trust center backup key for use with the **BK** command.

It is highly recommended to set **KB** prior to any network formation. See Centralized trust center backup for best practices.

USAGE:

KB

When sent without any parameters, **KB** returns **0** if no key has been set, otherwise it returns **OK**.

KB <New Key>

When one parameter is passed, this value is set as the key. As soon as the key is set in this fashion, **KY** and **NK** are cleared and an immediate **WR** is performed. This action is necessary to protect against the possibility of an unauthorized user changing this key and generating a backup in an attempt to glean sensitive information.

WARNING! This will invalidate the current network and require all devices reassociate after the network is reformed.

KB <Old Key> <New Key>

When two parameters are passed, if the first value matches the value set for either **KY** or **KB**, then **KB** is updated to the second value without clearing **KY** and **NK**. This causes no disruption to the existing network.

To protect against the possibility that an unauthorized user could attempt a brute-force attack, 20 invalid attempts to change **KB** in this manner will result in **KY**, **NK**, and **KB** being cleared and an immediate **WR** performed. These attempts persist across power cycles.

Parameter range

Up to a 256-bit value

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 Zigbee 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.

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 Zigbee 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 Zigbee RF interface of the device.

PL (TX Power Level)

Sets or displays the power level at which the device transmits conducted power for Zigbee 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

SC (Scan Channels)

The channels used when an active scan is performed by the local device.

An active scan is performed any time a network is formed or prior to a join attempt. You can force an active scan by issuing an **AS** command.

Changing **SC** after network association may cause the device to leave the network if the operating channel (**CH**) is excluded from the **SC** mask.

Parameter range

0 - 0xFFFF (bit field)

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

0x7FFF (channels 11 through 25)

SD (Scan Duration)

Sets or displays the length of time the device will linger on a channel during an energy scan and active scan.

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.

Note **SD** influences the time the MAC listens for beacons or runs an energy scan on a given channel. The **SD** time is not an accurate estimate of the router/end device joining time requirements. Zigbee joining includes additional overhead comprising beacon processing on each channel, and sending a join request that extends the actual joining time.

Parameter range

0 - 7 (exponent)

Default

3

MAC diagnostics commands

The following commands provide Media Access Control diagnostic information.

EA (MAC ACK Failure Count)

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

DB (Last Packet RSSI)

This command reports the received signal strength of the last received RF data packet or APS acknowledgment. The **DB** command only indicates the signal strength of the last hop. It does not provide an accurate quality measurement for a multihop link.

The **DB** command value is measured in -dBm. For example, if **DB** returns 0x50, then the RSSI of the last packet received was -80 dBm. Set **DB** to 0 to clear the current value.

Parameter range

0 - 0xFF

Default

N/A

ED (Energy Detect)

Measures the detected energy on each IEEE 802.15.4 channel.

In Transparent mode (**AP** = **0**), a comma follows each value with the list ending with a carriage return. The values returned reflect the detected energy level in units of -dBm. Convert an **ED** response of 49, 3A, and so on, to decimal to become -73 dBm, -58 dBm, and so on.

ED accepts a parameter value which will increase the duration of the energy detection scan.

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.

When **SM** > **0**, the device operates as an end device. However, **CE** must be **0** before **SM** can be set to a value greater than **0** to change the device to an end device. Changing a device from a router to an end device (or vice versa) forces the device to leave the network and attempt to join as the new device type when changes are applied.

Parameter range

0, 1, 4, 5

Parameter	Description
0	Sleep disabled (router)
1	Pin sleep
2	N/A
3	N/A
4	Cyclic sleep enabled
5	Cyclic sleep, pin wake
6	MicroPython sleep (with optional pin wake). For complete details see the *Digi MicroPython Programming Guide*.

Default

0

SP (Cyclic Sleep Period)

Sets the duration of sleep time for the end device, up to 28 seconds. Use the **SN** command to extend the sleep time past 28 seconds.

On the parent, this value determines how long the parent buffers a message for the sleeping end device. Set the value to at least equal to the longest **SP** time of any child end device.

Parameter range

0x20 - 0xAF0 x 10 ms (Quarter second resolution)

Default

0x20

ST (Cyclic Sleep Wake Time)

Sets or displays the wake time of a cyclically sleeping end device after receiving serial or RF data.

The wake timer resets each time the device receives serial or RF data. Once the timer expires, an end device may enter low power operation.

Parameter range

1 - 0xFFFF (x 1 ms)

Default

0x1388 (5 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 - 0xFF

Bit field:

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

Bit	Option
0	Reserved.
1	Wake for the entire **ST** time per wake period.
2	Enable extended cyclic sleep (sleep for the entire **SN** * **SP** time, possible data loss).

Default

0

WH (Wake Host Delay)

Sets or displays the wake host timer value. You can use **WH** to give a sleeping host processor sufficient time to power up after the device asserts the ON_SLEEP line.

If you set **WH** to a non-zero value, this timer specifies a time in milliseconds that the device delays after waking from sleep before sending data out the UART or transmitting an I/O sample. If the device receives serial characters, the **WH** timer stops immediately.

Parameter range

0 - 0xFFFF (x 1 ms)

Default

0

PO (Polling Rate)

Set or read the end device poll rate.

Setting this to 0 (default) enables polling at 100 ms (default rate), advancing in 10 ms increments. Adaptive polling may allow the end device to poll more rapidly for a short time when receiving RF data.

Parameter range

0 - 0x3E8 (x 10 ms)

Default

0

ET (End Device Timeout)

Sets the child table parent timeout. This command is only set on the sleepy end device. The sleepy end device sends the timeout to the parent when joining the network.

Parameter range

0 - 14

Parameter	Child table timeout
0	10 seconds

Parameter	Child table timeout
1	2 minutes
2	4 minutes
3	8 minutes
4	16 minutes
5	32 minutes
6	64 minutes
7	128 minutes
8	256 minutes
9	512 minutes
10	1024 minutes
11	2048 minutes
12	4096 minutes
13	8192 minutes
14	16384 minutes

Default

1 (2 minutes)

SI (Sleep Immediately)

Executable command. Causes a cyclic sleep device to sleep immediately rather than wait for the **ST** timer to expire.

Note If you issue this command in Command mode, the module remains in Command mode until the **CT** timer expires or you issue a **CN** command.

Instructs a synchronously sleeping network to go to sleep before before **ST** expires. It begins with the node that receives the command and affects every node in the network that can hear the broadcast that requests the network to sleep immediately.

It is only effective if the network is in sleep compatibility node—**SM8** or **SM7**.

Parameter

N/A

Default

N/A

MicroPython commands

The following commands relate to using MicroPython on the XBee 3 Zigbee RF Module.

PS (Python Startup)

Sets whether or not the XBee 3 Zigbee 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 Zigbee 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.

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 Zigbee 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 Zigbee RF Module cannot be in Sleep mode. If the device is configured to allow Sleep mode and you enable Bluetooth, the XBee 3 Zigbee RF Module will not enter sleep.

Parameter range

Parameter	Description
0	Bluetooth functionality is disabled.
1	Bluetooth functionality is enabled.

Default

0

BL (Bluetooth 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 Zigbee**.

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 Zigbee 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 Zigbee 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 Zigbee 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 $S (SRP Salt) to create an encrypted password for the XBee 3 Zigbee RF Module. Together, **$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)

Determines the API mode for the UART interface.

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 Options)

Configure the serial output options for received API frames. This parameter is only applicable when the device is operating in API mode (**AP** = **1** or **2**) and will also affect the frames that are received through MicroPython via the **xbee.receive()** function. For more information about ZDO packet handling, see Receiving ZDO commands and responses.

- When **AO** is set to **0**, a basic 0x90 receive frame type will be emitted when data packets are received by the device. No ZDO messages are emitted when configured this way.
- When **AO** is non-zero, received data packets will be emitted as explicit 0x91 frames.
- **AO** bits 1, 2, and 3 determine the routing of received ZDO messages. By default, the XBee application will handle all received messages, but for supporting external Zigbee applications, the received messages can instead be passed through to the serial port by setting these bits.
- **AO** bit 4 will allow supported ZDO message that are handled by the XBee application to be echoed to the serial port.

Leave unused bits clear so future API options are not inadvertently enabled during a firmware update.

Bit field

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

Bit	Description
0	0 = Native API output (0x90 frame type) 1 = Explicit API output (0x91 frame type)
1	0 = XBee handles Supported ZDO requests 1 = Supported ZDO request pass-through
2	0 = XBee handles UnsupportedZDOrequests (responds with **ZDO not supported**) 1 = Unsupported ZDO request pass-through
3	0 = XBee handles Binding requests (responds with **ZDO not supported**) 1 = Binding request pass-through
4	This bit is only applicable when **AO** bit 1 = **0** (XBee handles incoming ZDO) 1 = Echo supported ZDO requests to the serial port
5	1 = Prevent any ZDO messages from going out the serial port. This will also disable any pass-through set by other **AO** bits.

Parameter range

0 - 0xFF

Default

0

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.

Parameter range

0x00 - 0x0A (bitfield)

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

Bit	Description
0	Suppress ZCL output
1	Output receive frames for FOTA update commands
2	Output supported ZCL packets
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

Parameter	Description
0x6	57600 b/s
0x7	115200 b/s
0x8	230,400 b/s
0x9	460,800 b/s
0xA	921,600 b/s

Default

0x03 (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

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 - 0x28F

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 - 0xCE4 (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)

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
3	Digital input
4	Digital output, low
5	Digital output, high

Default

0

D7 (DIO7/CTS)

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
3	Digital input
4	Digital output, low
5	Digital output, high
6	RS-485 enable, low Tx
7	RS-485 enable, high Tx

Default

1

P3 (DIO13/DOUT Configuration)

Sets or displays the DIO13/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
4	Digital output, low
5	Digital output, high

Default

1

P4 (DIO14/DIN Configuration)

Sets or displays the DIO14/DIN configuration (Micro pin 4/SMT pin 4/TH pin 3).

Parameter range

0, 1, 3 - 5

Parameter	Description
0	Disabled

Parameter	Description
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 configuration (Micro pin 16/SMT pin 17/TH Pin N/A).

Note The DIO15 configuration is not available with the XBee 3 through-hole module.

Parameter range

0, 1, 4, 5

Parameter	Description
0	Disabled
1	SPI_MISO
4	Digital output, low
5	Digital output, high

Default

1

P6 (DIO16/SPI_MOSI Configuration)

Sets or displays the DIO16 configuration (Micro pin 15/SMT pin 16/TH Pin N/A).

Note The DIO16 configuration is not available with the XBee 3 through-hole module.

Parameter range

0, 1, 4, 5

Parameter	Description
0	Disabled
1	SPI_MOSI
4	Digital output, low
5	Digital output, high

Default

1

P7 (DIO17/SPI_SSEL Configuration)

Sets or displays the DIO17 configuration (Micro pin 14/SMT pin 15/TH Pin N/A).

Note The DIO17 configuration is not available with the XBee 3 through-hole module.

Parameter range

0, 1, 4, 5

Parameter	Description
0	Disabled
1	SPI_$\overline{\text{SSEL}}$
4	Digital output, low
5	Digital output, high

Default

1

P8 (DIO18/SPI_CLK Configuration)

Sets or displays the DIO18 configuration (Micro pin 13/SMT pin 14/TH Pin N/A).

Note The DIO18 configuration is not available with the XBee 3 through-hole module.

Parameter range

0, 1, 4, 5

Parameter	Description
0	Disabled
1	SPI_CLK
4	Digital output, low
5	Digital output, high

Default

1

P9 (DIO19/SPI_ATTN Configuration)

Sets or displays the DIO19 configuration (Micro pin 11/SMT pin 12/TH Pin N/A).

Note The DIO19 configuration is not available with the XBee 3 through-hole module.

Parameter range

0, 1, 4, 5

Parameter	Description
0	Disabled
1	SPI_$\overline{\text{ATTN}}$
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 Zigbee RF Module.

D0 (DIO0/AD0/Commissioning Button Configuration)

Sets or displays the DIO0/AD0/CB configuration (Micro pin 31/SMT pin 33/TH pin 20).

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 Pushbutton)

Use **CB** to simulate Commissioning Pushbutton presses in software.

You can issue **CB** even if the Commissioning Button functionality is disabled (**D0** is not set to **1**).

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, 2, 4

Parameter	Description
1	If disassociated: ■ Join Network. If associated: ■ Wake device for 30 seconds, if sleeping. ■ Send Node Identification broadcast.
2	Enable joining for 1 minute (or **NJ** seconds if **NJ** is not 0 or 0xFF).
4	Restore device configuration to default and leave the network.

Default

N/A

D1 (AD1/DIO1/TH_SPI_ATTN Configuration)

Sets or displays the DIO1/AD1 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_$\overline{\text{ATTN}}$ for the through-hole device N/A for the surface-mount device
2	ADC
3	Digital input
4	Digital output, low
5	Digital output, high

Default

0

D2 (DIO2/AD2/TH_SPI_CLK Configuration)

Sets or displays the DIO2/AD2 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
3	Digital input
4	Digital output, low
5	Digital output, high

Default

0

D3 (DIO3/AD3/TH_SPI_SSEL Configuration)

Sets or displays the DIO3/AD3 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 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 device
3	Digital input
4	Digital output, low
5	Digital output, high

Default

0

D5 (DIO5/Associate Configuration)

Sets or displays the DIO5 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
3	Digital input
4	Digital output, default low
5	Digital output, default high

Default

1

D8 (DIO8/DTR/SLP_RQ)

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
1	$\overline{\text{DTR}}$/Sleep Request (used with pin sleep and cyclic sleep with pin wake)
3	Digital input
4	Digital output, low
5	Digital output, high

Default

1

D9 (DIO9/ON_SLEEP)

Sets or displays the DIO9/ON_$\overline{\text{SLEEP}}$ configuration (Micro pin 25/SMT pin 26/TH pin 13).

Parameter range

0, 1, 3 - 5

Parameter	Description
0	Disabled
1	Awake/$\overline{\text{SLEEP}}$ indicator
3	Digital input
4	Digital output, low
5	Digital output, high

Default

1

P0 (DIO10/RSSI Configuration)

Sets or displays the DIO10/RSSI configuration (Micro pin 7/SMT pin 7/TH pin 6).

Parameter range

0 - 5

Parameter	Description
0	Disabled
1	RSSI PWM output

Parameter	Description
2	PWM0 output. M0 (PWM0 Duty Cycle) controls the value.
3	Digital input
4	Digital output, low
5	Digital output, high

Default

1

P1 (DIO11 Configuration)

Sets or displays the DIO11 configuration (Micro pin 8/SMT pin 8/TH pin 7).

Parameter range

0, 2 - 5

Parameter	Description
0	Disabled
1	N/A
2	PWM1 output. M1 (PWM1 Duty Cycle) controls 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 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

Parameter	Description
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 resistor status for the I/O lines.

- If you set a **PR** bit to 1, it enables the pull-up/down resistor
- If you set a **PR** bit to 0, it specifies no internal pull-up/down resistor.

PR and **PD** only affect lines that are configured as digital inputs (**3**) or disabled (**0**).
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
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

Bit	I/O line	Micro pin	Surface-mount pin	Through-hole pin
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

PD (Pull Up/Down Direction)

The resistor pull direction bit field (1 = pull-up, 0 = pull-down) for corresponding I/O lines that are set by the **PR** command.

If the bit is set, the device uses an internal pull-up resistor. If it is clear, the device uses an internal pull-down resistor. See the **PR** command for the bit order.

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/TH pin 6).

If P0 (DIO10/RSSI Configuration) is configured as PWM0 output, you can 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)

The duty cycle of the PWM1 line (Micro pin 8/SMT pin 8/TH pin 7).

If P1 (DIO11 Configuration) is configured as PWM1 output, you can 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 signal duty cycle updates with each received packet and shuts off when the timer expires.

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.

If **LT** = **0**, the device uses the default blink rate of 250 ms.

For all other **LT** values, the firmware measures **LT** in 10 ms increments.

Parameter range

0, 0xA - 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.

IR (I/O Sample Rate)

Determines the I/O sample rate used to generate outgoing I/O sample data. When the IR value is greater than 0, the device samples and transmits all enabled digital I/O and ADCs every **IR** milliseconds. I/O Samples transmit to the address specified by **DH** +**DL**.

At least one I/O line must be configured as an input or explicit output for samples to be generated.

Parameter range

0, 0x32 - 0xFFFF (ms)

Default

0

IC (Digital Change Detection)

The bit field that configures which digital I/O pins should be monitored for digital change detection. 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**.

Change Detect is edge-triggered and must occur while the device is awake. If the level transition occurs during a sleep period, the device will not see a change.

Bit field

Bit	I/O line	Micro pin	Surface-mount pin	Through-hole pin
0	DIO0	31	33	20
1	DIO1	30	32	19
2	DIO2	29	31	18
3	DIO3	28	30	17
4	DIO4	23	24	11
5	DIO5	26	28	15
6	DIO6	27	29	16
7	DIO7	24	25	12
8	DIO8	9	10	9
9	DIO9	25	26	13
10	DIO10	7	7	6
11	DIO11	8	8	7

Bit	I/O line	Micro pin	Surface-mount pin	Through-hole pin
12	DIO12	5	5	4
13	DIO13	3	3	2
14	DIO14	4	4	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

IS (Force Sample)

Immediately forces an I/O sample to be generated. If you issue the command to the local device, the sample data is sent out the local serial interface. If sent remotely, the sample data is returned as a 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.

Parameter range

N/A

Default

N/A

V+ (Supply Voltage Threshold)

Define the supply voltage threshold that appends the supply voltage to outgoing I/O sample frames.

If the measured supply voltage falls below or equal to this threshold, the supply voltage will be appended to outgoing I/O sample frames and set bit 7 of the Analog Channel Mask.

Set **V+** to **0** to not include the supply voltage.

Set **V+** to **1** to always include the supply voltage.

Example

To include a measurement of the supply voltage when it falls below 2.7 V, set **V+** to **2700** = **0xA8A**.

Parameter range

0 - 0xFFFF (in mV)

Default

0

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

0x1000 - 0xFFFF [read-only]

Default

Set in the firmware

VL (Version Long)

Shows detailed version information including the application build date and time.

Parameter range

Multi-line string [read-only]

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

%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

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

%V (Voltage Supply Monitoring)

Reads the voltage on the Vcc pin in mV.

Parameter range

0 - 0xFFFF (in mV) [read only]

Default

N/A

TP (Temperature)

The current module temperature in degrees Celsius. The temperature is represented in two's complement, as shown in the following example:

1 °C = 0x0001 and -1°C = 0xFFFF

Parameter range

0 - 0xFFFF (Celsius) [read-only]

Default

N/A

CK (Configuration Checksum)

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 and also applies queued command parameter values set with 0x09 API queued command frames.

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 all device parameters—except **ZS**, **C8**, and **KB**—to factory defaults but do not apply the parameters.

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 Zigbee 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

API Operation

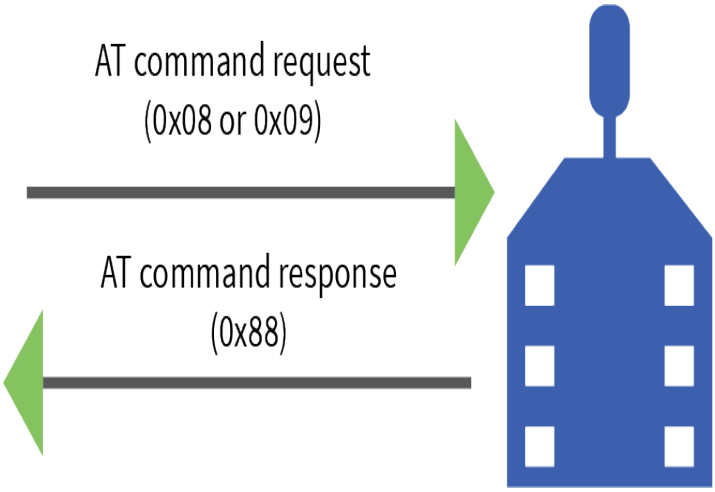
An alternative to Transparent Operation are Application Programming Interface (API) Operations. API operation requires that the device communicate through a structured interface (that is, data is communicated in frames in a defined order). The API specifies how the device sends and receives commands, command responses, and module status messages using a serial port Data Frame.

API serial exchanges

You can use the Frame ID field to correlate between the outgoing frames and associated responses.

AT commands

The following image shows the API frame exchange that takes place at the serial interface when sending an AT command request to read or set a device parameter. You can disable the response by setting the frame ID to 0 in the request.



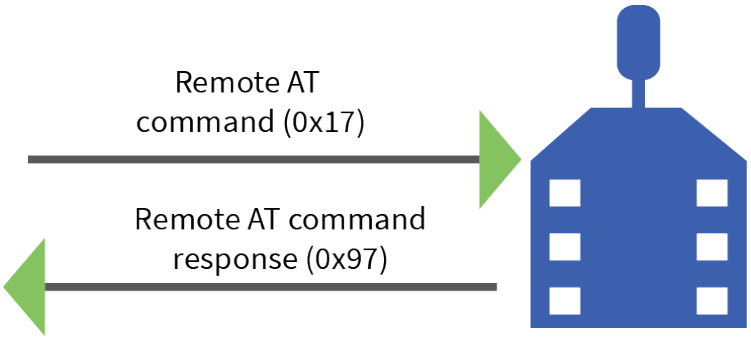
Transmit and Receive RF data

The following image shows the API frames exchange that take place at the UART interface when sending RF data to another device. The transmit status frame is always sent at the end of a data transmission unless the frame ID is set to 0 in the TX request. If the packet cannot be delivered to the destination, the transmit status frame indicates the cause of failure.

The received data frame type (0x90 or 0x91) is determined by the **AO** command.

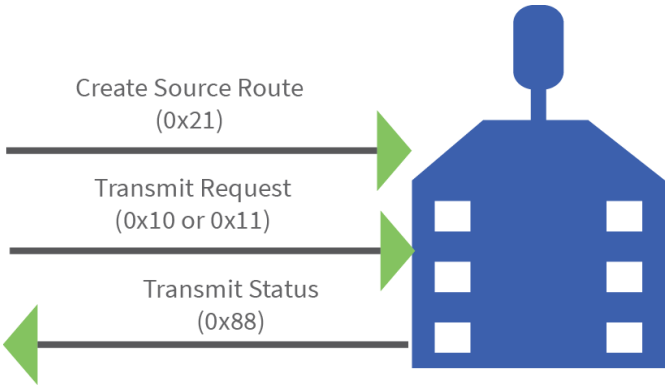
Remote AT commands

The following image shows the API frame exchanges that take place at the serial interface when sending a remote AT command. The device does not send out a remote command response frame through the serial interface if the remote device does not receive the remote command.



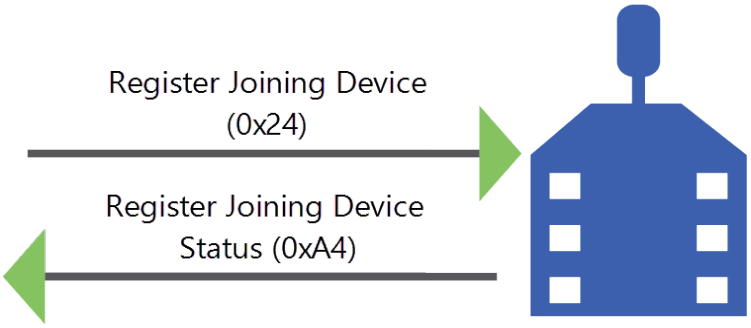
Source routing

The following image shows the API frame exchanges that take place at the serial port when sending a source routed transmission.



Device Registration

The following image shows the API frame exchanges that take place at the serial interface when registering a joining device to a trust center.



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 test data integrity, the device calculates and verifies a checksum on non-escaped data.

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 08 08 01 4E 49 58 42 45 45 3B**

Byte(s)	Description
7E	Start delimiter
00 08	Length bytes
08	API identifier
01	API frame ID
4E 49	AT Command
58 42 45 45	Parameter value
3B	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 08 08 **01 4E 49 58 42 45** 45 3B

Add these hex bytes:

0x08 + 0x01 + 0x4E + 0x49 + 0x58 + 0x42 + 0x45 + 0x45 = 0x01C4

Now take the result of 0x01C4 and keep only the lowest 8 bits which in this example is 0xC4 (the two far right digits). Subtract 0xC4 from 0xFF and you get 0x3B (0xFF - 0xC4 = 0x3B). 0x3B is the checksum for this data packet.

If an API data packet is composed with an incorrect checksum, the XBee 3 Zigbee 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.

0x08 + 0x01 + 0x4E + 0x49 + 0x58 + 0x42 + 0x45 + 0x45 + 0x3B = 0x01FF

Send ZDO commands with the API

Zigbee specifications define Zigbee device objects (ZDOs) as part of the Zigbee device profile. These objects provide functionality to manage and map out the Zigbee network and to discover services on Zigbee devices. ZDOs are typically required when developing a Zigbee product that interoperates in a public profile such as home automation or smart energy, or when communicating with Zigbee devices from other vendors. You can also use the ZDO to perform several management functions such as frequency agility (energy detect and channel changes - Mgmt Network Update Request), discovering routes (Mgmt Routing Request) and neighbors (Mgmt LQI Request), and managing device connectivity (Mgmt Leave and Permit Join Request).

The following table shows some of the more prominent ZDOs with their respective cluster identifier. Each ZDO command has a defined payload. See the *Zigbee device profile* section of the Zigbee specification for details.

ZDO command	Cluster ID
Network Address Request	0x0000
IEEE Address Request	0x0001
Node Descriptor Request	0x0002

ZDO command	Cluster ID
Simple Descriptor Request	0x0004
Active Endpoints Request	0x0005
Match Descriptor Request	0x0006
Mgmt LQI Request	0x0031
Mgmt Routing Request	0x0032
Mgmt Leave Request	0x0034
Mgmt Permit Joining Request	0x0036
Mgmt Network Update Request	0x0038

Use the Explicit Addressing Command Request - 0x11 to send Zigbee device objects commands to devices in the network. Sending ZDO commands with the Explicit Transmit API frame requires some formatting of the data payload field.

When sending a ZDO command with the API, all multiple byte values in the ZDO command (API payload), for example, u16, u32, and 64-bit addresses, must be sent in little endian byte order for the command to be executed correctly on a remote device.

For an API XBee to receive ZDO responses, set AO (API Options) to **1** to enable the explicit receive API frame.

The following table shows how you can use the Explicit API frame to send an “Active Endpoints” request to discover the active endpoints on a device with a 16-bit address of 0x1234.

Frame data fields	Offset	Description
Frame type	3	0x11
Frame ID	4	Identifies the data frame for the host to correlate with a subsequent transmit status. If set to **0**, the device does not send a response out the serial port.
64-bit destination address	5-12	MSB first, LSB last. The 64-bit address of the destination device (big endian byte order). For unicast transmissions, set to the 64-bit address of the destination device, or to 0x0000000000000000 to send a unicast to the coordinator. Set to 0x000000000000FFFF for broadcast.
16-bit destination network address	13	MSB first, LSB last. The 16-bit address of the destination device (big endian byte order). Set to 0xFFFE for broadcast, or if the 16-bit address is unknown.
	14	
Source endpoint	15	Set to 0x00 for ZDO transmissions (endpoint 0 is the ZDO endpoint).
Destination endpoint	16	Set to 0x00 for ZDO transmissions (endpoint 0 is the ZDO endpoint).

Frame data fields	Offset	Description
Cluster ID	17	Set to the cluster ID that corresponds to the ZDO command being sent. 0x0005 = Active Endpoints Request
	18	
Profile ID	19-20	Set to 0x0000 for ZDO transmissions (Profile ID 0x0000 is the Zigbee device profile that supports ZDOs).
Broadcast radius	21	Sets the maximum number of hops a broadcast transmission can traverse. If set to **0**, the device sets the transmission radius to the network maximum hops value.
Transmission options	22	All bits must be set to 0.
Data payload	23	The required payload for a ZDO command. All multi-byte ZDO parameter values (u16, u32, 64- bit address) must be sent in little endian byte order. The Active Endpoints Request includes the following payload: [16-bit NwkAddrOfInterest] **Note** The 16-bit address in the API example (0x1234) is sent in little endian byte order (0x3412).
	24	
	25	

Example

The following example shows how you can use the Explicit API frame to send an “Active Endpoints” request to discover the active endpoints on a device with a 16-bit address of 0x1234.

Frame data fields	Offset	Example
Start delimiter	0	0x7E
Length	MSB 1	0x00
	LSB 2	0x17
Frame type	3	0x11
Frame ID	4	0x01
64-bit destination address	MSB 5	0x00
	6	0x00
	7	0x00
	8	0x00
	9	0x00
	10	0x00
	11	0xFF
	LSB12	0xFF

Frame data fields	Offset	Example
16-bit Destination Network Address	MSB 13	0xFF
	LSB 14	0xFE
Source endpoint	15	0x00
Destination endpoint	16	0x00
Cluster ID	17	0x00
	18	0x05
Profile ID	19	0x00
	20	0x00
Broadcast radius	21	0x00
Transmit options	22	0x00
Data payload - transaction sequence number	23	0x01
Data payload - ZDO payload	24	0x34
	25	0x12
Checksum	29	0xA6

Send Zigbee cluster library (ZCL) commands with the API

The Zigbee cluster library defines a set of attributes and commands (clusters) that can be supported in multiple Zigbee profiles. The ZCL commands are typically required when developing a Zigbee product that will interoperate in a public profile such as home automation or smart energy, or when communicating with Zigbee devices from other vendors. Applications that are not designed for a public profile or for interoperability applications can skip this section.

The following table shows some prominent clusters with their respective attributes and commands.

Cluster (Cluster ID)	Attributes (Attribute ID)	Cluster ID
Basic (0x0000)	Application Version (0x0001) Hardware Version (0x0003) Model Identifier (0x0005)	Reset to defaults (0x00)
Identify (0x0003)	Identify Time (0x0000)	Identify (0x00) Identify Query (0x01)
Time (0x000A)	Time (0x0000) Time Status (0x0001) Time Zone (0x0002)	
Thermostat (0x0201)	Local Temperature (0x0000) Occupancy (0x0002)	Setpoint raise / lower (0x00)

The ZCL defines a number of profile-wide commands that can be supported on any profile, also known as general commands. These commands include the following.

Command (Command ID)	Description
Read Attributes (0x00)	Used to read one or more attributes on a remote device.
Read Attributes Response (0x01)	Generated in response to a read attributes command.
Write Attributes (0x02)	Used to change one or more attributes on a remote device.
Write Attributes Response (0x04)	Sent in response to a write attributes command.
Configure Reporting (0x06)	Used to configure a device to automatically report on the values of one or more of its attributes.
Report Attributes (0x0A)	Used to report attributes when report conditions have been satisfied.
Discover Attributes (0x0C)	Used to discover the attribute identifiers on a remote device.
Discover Attributes Response (0x0D)	Sent in response to a discover attributes command.

Use the Explicit Addressing Command Request - 0x11 to send ZCL commands to devices in the network. Sending ZCL commands with the Explicit Transmit API frame requires some formatting of the data payload field.

When sending a ZCL command with the API, all multiple byte values in the ZCL command (API Payload) (for example, u16, u32, 64-bit addresses) must be sent in little endian byte order for the command to be executed correctly on a remote device.

Note When sending ZCL commands, set the AO command to 1 to enable the explicit receive API frame. This provides indication of the source 64- and 16-bit addresses, cluster ID, profile ID, and endpoint information for each received packet. This information is required to properly decode received data.

The following table shows how the Explicit API frame can be used to read the hardware version attribute from a device with a 64-bit address of 0x0013A200 40401234 (unknown 16-bit address). This example uses arbitrary source and destination endpoints. The hardware version attribute (attribute ID 0x0003) is part of the basic cluster (cluster ID 0x0000). The Read Attribute general command ID is 0x00.

Frame fields			Offset	Description
Frame type			3	
Frame ID			4	Identifies the serial port data frame for the host to correlate with a subsequent transmit status. If set to 0, no transmit status frame will be sent out the serial port.

Frame fields			Offset	Description
64-bit destination address			MSB 5	The 64-bit address of the destination device (big endian byte order). For unicast transmissions, set to the 64-bit address of the destination device, or to 0x0000000000000000 to send a unicast to the coordinator. Set to 0x000000000000FFFF for broadcast.
			6	
			7	
			8	
			9	
			10	
			11	
			LSB 12	
16-bit destination network address			MSB 13	The 16-bit address of the destination device (big endian byte order). Set to 0xFFFE for broadcast, or if the 16-bit address is unknown.
			LSB 14	
Source endpoint			15	Set to the source endpoint on the sending device (0x41 arbitrarily selected).
Destination endpoint			16	Set to the destination endpoint on the remote device (0x42 arbitrarily selected).
Cluster ID			MSB 17	Set to the cluster ID that corresponds to the ZCL command being sent. 0x0000 = Basic Cluster.
			LSB 18	
Profile ID			MSB 19	Set to the profile ID supported on the device (0xD123 arbitrarily selected).
			LSB 20	
Broadcast radius			21	Sets the maximum number of hops a broadcast transmission can traverse. If set to 0, the transmission radius will be set to the network maximum hops value.
Transmit options			22	All bits must be set to 0.
Data payload	ZCL frame header	Frame control	23	Bitfield that defines the command type and other relevant information in the ZCL command. For more information, see the ZCL specification.
		Transaction sequence number	24	A sequence number used to correlate a ZCL command with a ZCL response. (The hardware version response will include this byte as a sequence number in the response.) The value 0x01 was arbitrarily selected.

Frame fields			Offset	Description
		Command ID	25	Since the frame control “frame type” bits are 00, this byte specifies a general command. Command ID 0x00 is a Read Attributes command.
	ZCL payload	Attribute ID	26	The payload for a “Read Attributes” command is a list of Attribute Identifiers that are being read. **Note** The 16-bit Attribute ID (0x0003) is sent in little endian byte order (0x0300). All multi- byte ZCL header and payload values must be sent in little endian byte order.
			27	0xFF minus the 8 bit sum of bytes from offset 3 to this byte.

Example

In this example, the Frame Control field (offset 23) is constructed as follows:

Name	Bits	Example Value Description
Frame Type	0-1	00 - Command acts across the entire profile.
Manufacturer Specific	2	0 - The manufacturer code field is omitted from the ZCL Frame Header.
Direction	3	0 - The command is being sent from the client side to the server side.
Disable Default Response	4	0 - Default response not disabled.
Reserved	5-7	Set to 0.

For more information, see the *Zigbee Cluster Library* specification.

Frame data fields			Offset	Example
Start delimiter			0	0x7E
Length			MSB 1	0x00
			LSB 2	0x19
Frame type			3	0x11
Frame ID			4	0x01

Frame data fields			Offset	Example
64-bit destination address			MSB 5	0x00
			6	0x13
			7	0xA2
			8	0x00
			9	0x40
			10	0x40
			11	0x12
			LSB12	0x34
16-bit destination network address			MSB 13	0xFF
			LSB 14	0xFE
Source endpoint			15	0x41
Destination endpoint			16	0x42
Cluster ID			MSB 17	0x00
			LSB 18	0x00
Profile ID			MSB 19	0xD1
			LSB 20	0x23
Broadcast radius			21	0x00
Transmit options			22	0x00
Data payload	ZCL frame header	Frame control	23	0x00
		Transaction sequence number	24	0x01
		Command ID	25	0x00
	ZCL payload	Attribute ID	26	0x03
			27	0x00
Checksum			28	0xFA

Send Public Profile Commands with the API

You can use the XBee API using the Explicit Transmit API frame (0x11) to send commands in public profiles such as Smart Energy and Home Automation. Sending public profile commands with the Explicit Transmit API frame requires some formatting of the data payload field. Most of the public profile commands fit into the Zigbee cluster library (ZCL) architecture as described in Send Zigbee cluster library (ZCL) commands with the API.

The following table shows how you can use the Explicit API frame to send a demand response and load control message (cluster ID 0x701) in the smart energy profile (profile ID 0x0109) in the revision 14 Smart Energy specification. The device sends a “Load Control Event”message (command ID 0x00) and to a device with 64- bit address of 0x0013A200 40401234 with a 16-bit address of 0x5678. The event starts a load control event for water heaters and smart appliances for a duration of 1 minute, starting immediately.

Note When sending public profile commands, set the **AO** command to 1 to enable the explicit receive API frame. This provides indication of the source 64- and 16-bit addresses, cluster ID, profile ID, and endpoint information for each received packet. This information is required to properly decode received data.

Frame specific data

Frame Fields			Offset	Description
Frame type			3	
Frame ID			4	Identifies the serial port data frame for the host to correlate with a subsequent transmit status. If set to 0, no transmit status frame will be sent out the serial port.
64-bit destination address			MSB 5	The 64-bit address of the destination device (big endian byte order). For unicast transmissions, set to the 64-bit address of the destination device, or to 0x0000000000000000 to send a unicast to the coordinator. Set to 0x000000000000FFFF for broadcast.
			6	
			7	
			8	
			9	
			10	
			11	
			LSB 12	
16-bit destination network address			MSB 13	The 16-bit address of the destination device (big endian byte order). Set to 0xFFFE for broadcast, or if the 16-bit address is unknown.
			LSB 14	
Source endpoint			15	Set to the source endpoint on the sending device. (0x41 arbitrarily selected).

Frame Fields			Offset	Description
Destination endpoint			16	Set to the destination endpoint on the remote device. (0x42 arbitrarily selected).
Cluster ID			MSB 17	Set to the cluster ID that corresponds to the ZCL command being sent. 0x0701 = Demand response and load control cluster ID
			LSB 18	
Profile ID			MSB 19	Set to the profile ID supported on the device. 0x0109 = Smart Energy profile ID.
			LSB 20	
Broadcast radius			21	Sets the maximum number of hops a broadcast transmission can traverse. If set to 0, the transmission radius will be set to the network maximum hops value.
Transmit options			22	All bits must be set to 0.
Data payload	ZCL frame header	Frame control	23	Bitfield that defines the command type and other relevant information in the ZCL command. For more information, see the ZCL specification.
		Transaction sequence number	24	A sequence number used to correlate a ZCL command with a ZCL response. (The hardware version response will include this byte as a sequence number in the response.) The value 0x01 was arbitrarily selected.
			25	Since the frame control “frame type” bits are 01, this byte specifies a cluster-specific command. Command ID 0x00 in the Demand Response and Load Control cluster is a Load Control Event command. For more information, see the Smart Energy specification.
	ZCL payload - load control event data	Issuer event ID	26	The 4-byte unique identifier. **Note** The 4-byte ID is sent in little endian byte order (0x78563412). The event ID in this example (0x12345678) is arbitrarily selected.
			27	
			28	
			29	

Frame Fields			Offset	Description
		Device class	30	This bit encoded field represents the Device Class associated with the Load Control Event. A bit value of 0x0014 enables smart appliances and water heaters. **Note** The 2-byte bit field value is sent in little endian byte order.
			31	
		Utility enrollment group	32	Used to identify sub-groups of devices in the device-class. 0x00 addresses all groups.
		Start time	33	
			34	
			35	
			36	
		Duration in minutes	37	This 2-byte value must be sent in little endian byte order.
			38	
		Criticality level	39	Indicates the criticality level of the event. In this example, the level is “voluntary”.
		Cooling temperature	40	Requested offset to apply to the normal cooling set point. A value of 0xFF indicates the temperature offset value is not used.
		Heating temperature offset	41	Requested offset to apply to the normal heating set point. A value of 0xFF indicates the temperature offset value is not used.
		Cooling temperature set point	42	Requested cooling set point in 0.01 degrees Celsius. A value of 0x8000 means the set point field is not used in this event. **Note** The 0x80000 is sent in little endian byte order.
			43	

Frame Fields			Offset	Description
		Heating temperature set point	44	Requested heating set point in 0.01 degrees Celsius. A value of 0x8000 means the set point field is not used in this event. **Note** The 0x80000 is sent in little endian byte order.
			45	
		Average load adjustment percentage	46	Maximum energy usage limit. A value of 0x80 indicates the field is not used.
		Duty cycle	47	Defines the maximum “On” duty cycle. A value of 0xFF indicates the duty cycle is not used in this event.
		Duty cycle event control	48	A bitmap describing event options.

Example

In this example, the Frame Control field (offset 23) is constructed as follows:

Name	Bits	Example Value Description
Frame Type	0-1	01 - Command is specific to a cluster
Manufacturer Specific	2	0 - The manufacturer code field is omitted from the ZCL Frame Header.
Direction	3	1 - The command is being sent from the server side to the client side.
Disable Default Response	4	0 - Default response not disabled
Reserved	5-7	Set to 0.

For more information, see the Zigbee cluster library specification.

Frame fields			Offset	Example
Start delimiter			0	0x7E
Length			MSB 1	0x00
			LSB 2	0x19
Frame type			3	0x11
Frame ID			4	0x01

Frame fields			Offset	Example
64-bit destination address			MSB 5	0x00
			6	0x13
			7	0xA2
			8	0x00
			9	0x40
			10	0x40
			11	0x12
			LSB 12	0x34
16-bit destination network address			MSB 13	0x56
			LSB 14	0x78
Source endpoint			15	0x41
Destination endpoint			16	0x42
Cluster ID			MSB 17	0x07
			LSB 18	0x01
Profile ID			MSB 19	0x01
			LSB 20	0x09
Broadcast radius			21	0x00
Transmit options			22	0x00

Frame fields			Offset	Example
Data payload	ZCL frame header	Frame control	23	0x09
		Transaction sequence number	24	0x01
			25	0x00
	ZCL payload - load control event data	Issuer event ID	26	0x78
			27	0x56
			28	0x34
			29	0x12
		Device class	30	0x14
			31	0x00
		Utility enrollment group	32	0x00
		Start time	33	0x00
			34	0x00
			35	0x00
			36	0x00
		Duration in Minutes	37	0x01
			38	0x00
		Criticality level	39	0x04
		Cooling temperature	40	0xFF
		Heating temperature offset	41	0xFF
		Cooling temperature set point	42	0x00
			43	0x80
		Heating temperature set point	44	0x00
			45	0x80
		Average load adjustment percentage	46	0x80
		Duty cycle	47	0xFF
		Duty cycle event control	48	0x00
Checksum			49	0x5B

Frame descriptions

The following sections describe the API frames.

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 RF payload size 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**

Zigbee-specific addressing information

- A Zigbee coordinator can be addressed in one of two ways:
  - Set the 64-bit address to all 0x00s and the 16-bit address to**0xFFFE**
  - Set the 64-bit address to the coordinator's 64-bit address and the 16-bit address to**0x0000**
- When using 64-bit addressing, populating the correct 16-bit address of the destination helps improve performance when transmitting to multiple devices. If you do not know a 16-bit address, set this field to**0xFFFE**(unknown). If the transmission is successful, the Extended Transmit Status - 0x8B indicates the discovered 16-bit address.
- When using 16-bit addressing, the following addresses are reserved:
  - **0xFFFC** = Broadcast to all routers
  - **0xFFFD** = Broadcast to all non-sleepy devices
  - **0xFFFF** = Broadcast to all devices, including sleepy end devices

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**.

Zigbee

Bit	Meaning	Description
0	Disable ACK [**0x01**]	Disable retries and route repair
1	Reserved	<set this bit to 0>
2	Indirect Transmission [**0x04**]	Used for Binding transmissions.
3	Multicast [**0x08**]	See Multicast transmissions for more information.

Bit	Meaning	Description
4	Secure Session Encryption [**0x10**]	Encrypt payload for transmission across a Secure Session Reduces maximum payload size by 4 bytes.
5	Enable APS encryption [**0x20**]	APS encrypt the payload using the link key set by **KY** Reduces maximum payload size by 4 bytes.
6	Use extended timeout [**0x40**]	See Extended timeout for more information.

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	0x0013A200 12345678	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 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	0x00000000 0000FFFF	0xFFFE	0x01	0x00	0x42726F616463617374
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 RF payload size 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**

Zigbee-specific addressing information

- A Zigbee coordinator can be addressed in one of two ways:
  - Set the 64-bit address to all 0x00s and the 16-bit address to **0xFFFE**
  - Set the 64-bit address to the coordinator's 64-bit address and the 16-bit address to **0x0000**
- When using 64-bit addressing, populating the correct 16-bit address of the destination helps improve performance when transmitting to multiple devices. If you do not know a 16-bit address, set this field to **0xFFFE** (unknown). If the transmission is successful, the Extended Transmit Status - 0x8B indicates the discovered 16-bit address.
- When using 16-bit addressing, the following addresses are reserved:
  - **0xFFFC** = Broadcast to all routers
  - **0xFFFD** = Broadcast to all non-sleepy devices
  - **0xFFFF** = Broadcast to all devices, including sleepy end devices

- To send a ZDO command (ZCL/ZDP):
  - Enter the ZDO command in the command data field (payload).
  - Each field in the ZDO command frame is represented in little endian format.
  - For information on the command formatting, refer to the ZCL and ZDP specifications.

Reserved endpoints

For serial data transmissions, the **0xE8** endpoint should be used for both source and destination endpoints.

The active Digi endpoints are:

- **0xE8** - Digi Data endpoint
- **0xE6** - Digi Device Object (DDO) endpoint
- **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.

Offset	Size	Frame Field	Description
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**.

Zigbee

Bit	Meaning	Description
0	Disable ACK [**0x01**]	Disable retries and route repair
1	Reserved	<set this bit to 0>
2	Indirect Transmission [**0x04**]	Used for Binding transmissions.
3	Multicast [**0x08**]	See Multicast transmissions for more information.
4	Secure Session Encryption [**0x10**]	Encrypt payload for transmission across a Secure Session Reduces maximum payload size by 4 bytes.
5	Enable APS encryption [**0x20**]	APS encrypt the payload using the link key set by **KY** Reduces maximum payload size by 4 bytes.
6	Use extended timeout [**0x40**]	See Extended timeout for more information.

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	0x5478446174 61
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 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	0000000 00 0000FF FF	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	0x0013A200 12345678	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*

Create Source Route - 0x21

Description

This frame type is used to create an entry in the source route table of a local device. A source route specifies the complete route a packet traverses to get from source to destination. For best results, use source routing with many-to-one routing. See Source routing for more information.

In most cases, this frametype is used in combination with routing information received from a corresponding Route Record Indicator - 0xA1 frame. Route indicators are generated when a network device sends data to a concentrator. The order in which addresses are entered into the 0x21 frame are the same as provided by the 0xA1 frame—destination to source.

There is no response frame for this frame type. Take care when generating source routes as an incorrectly formatted frame will be silently rejected or may cause unexpected results.

Note Both the 64-bit and 16-bit destination addresses are required when creating a source route.

Format

The following table provides the contents of the frame. For details on frame structure, seeAPI 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**	Create Source Route - **0x21**
4	8-bit	**Frame ID (reserved)**	This frame type generates no response, so the Frame ID field is not used. Set this field to 0.
5	64-bit	**64-bit destination address**	Set to the 64-bit IEEE address of the destination device (required).
13	16-bit	**16-bit destination address**	Set to the 16-bit network address of the destination device (required).
15	8-bit	**Options (reserved)**	Source routing options are not available yet. This bit field is reserved for future functionality. Set this field to 0.

Offset	Size	Frame Field	Description
16	8-bit	**Number of addresses**	The number of addresses in the source route (excluding source and destination). A route can only traverse across a maximum of 30 hops. If this number is **0** or exceeds the maximum hop count, the frame is silently discarded and a route will not be created.
17-n	16-bit variable	**Address**	The 16-bit network address(es) of the devices along the source route, excluding the source and destination. The addresses should be entered in reverse order (from destination to source) to match the order provided in Route Record Indicator - 0xA1.
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.

4-hop route

A concentrator needs to send data to a device with a 64-bit address of **0013A20012345678** that is 4-hops away. Due to the size of the network, route discoveries need to be be minimized. The concentrator had previously received a Route Record Indicator - 0xA1 with the source route to this device and had stored this information.

The route looks like this:

Source (concentrator) <> **Router A** <> **Router B** <> **Router C** <> **Destination (remote)**

In this example, the network addresses are simplified.

```
7E 00 14 21 00 00 13 A2 00 12 34 56 78 DD DD 00 03 CC CC BB BB AA AA F6
```

Frame type	Frame ID	64-bit dest	16-bit dest	Options	Num of addresses	Address 1	Address 2	Address 3
0x21	0x00	0x0013A200 12345678	0xDDDD	0x00	0x03	0xCCCC	0xBBBB	0xAAAA
Route request	*Not used*	*Destination IEEE address*	*Destination NWK address*	*Not used*		*Neighbor of dest*	*Intermediate hop*	*Neighbor of source*

Register Joining Device - 0x24

Response frame: Registration Status - 0xA4

Description

This frame type is used to securely register a joining device to a trust center. Registration is the process by which a node is authorized to join the network using a pre-configured or installation code-derived link key that is conveyed to the trust center out-of-band—using an interface that is not the Zigbee network.

If registering a device with a centralized trust center—**EO** = **2**—then the key entry will only persist for **KT** seconds before expiring, or until the device joins the network whereby the key is cleared.

Registering devices in a distributed trust center—**EO** = **0**—is persistent and the key entry will never expire unless explicitly removed. To remove a key entry on a distributed trust center, a 0x24 frame should be issued with a null key—key field is absent from the frame. In a centralized trust center you cannot use this method to explicitly remove the key entries before the **KT** timeout.

The registration frame will accept all 0xFF's for the device address (EUI) as a wild card. You can only register one entry at a time with the trust center in this manner. Only after the **KT** period expires can you enter an additional wildcard entry. We do not recommend this method. A best practice is to specifically set the EUI for every device individually.

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**	Register Joining Device - **0x24**
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 registrant address**	Set to the 64-bit IEEE address of the joining device. Set to **0xFFFFFFFFFFFFFFFF** to act as a one-time use wildcard.
13	16-bit	**Reserved**	Not used. Set to **0xFFFE**.

Offset	Size	Frame Field	Description
15	8-bit	**Options**	Bit field of options that apply to device registration: ■ **Bit 0**: Key type • [**0x00**] = **Pre-configured Link Key** - Register device using a pre-configured link key ◦ Key field is the link key—**KY** command—of the joining device. • [**0x01**] = **Install Code** - Register device using an installation code-derived link key. ◦ Key field is the install code—**I?** command—of the joining device. **Note** Option values may be combined. Set all unused bits to 0.
16-n	variable	**Key/Install Code**	When registering using a pre-configured link key, field accepts up to 16-bytes. When registering using an install code, enter the installation code + CRC—**I?** command—of the joining device. Up to 18-bytes, the CRC can be any endianness.
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.

Pre-configured link key registration

A device with the address of **0013A20012345678** needs to join a secured network using a pre-configured link key of **12345**. This link key is unknown to the trust center—**KY** does not match—thus it must be conveyed out-of-band via a registration frame.

The corresponding Registration Status - 0xA4 response with a matching Frame ID will indicate whether the registration succeeded.

```
7E 00 10 24 5D 00 13 A2 00 12 34 56 78 FF FE 00 01 23 45 4F
```

Frame type	Frame ID	64-bit reg address	Reserved	Options	Key/Install Code
0x24	0x5D	0x0013A200 12345678	0xFFFE	0x00	0x012345
Registration	*Matches response*		*N/A*	*Key is a link key*	*Pre-configured Link Key (**KY**)*

Installation code-derived link key registration

A device with the address of **0013A20012345678** needs to join a secured network using an install code of **620D28BDAF2A569B54E7377E33C504A099F1**. The install code read by the **I?** command includes the 2-byte CRC, the install code can be read from a device and entered into the frame as-is.

The corresponding Registration Status - 0xA4 response with a matching Frame ID will indicate whether the registration succeeded.

```
7E 00 1F 24 1C 00 13 A2 00 12 34 56 78 FF FE 01 62 0D 28 BD AF 2A 56 9B 54 E7
37 7E 33 C5 04 A0 99 F1 C4
```

Frame type	Frame ID	64-bit reg address	Reserved	Options	Key/Install Code
0x24	0x1C	0x0013A200 12345678	0xFFFE	0x01	0x620D28BDAF2A569B 54E7377E33C504A0 99F1
Registration	*Matches response*		*N/A*	*Key is an install code*	*Install code (**I?**)*

Distributed trust center: link key de-registration

A previously registered device with the 64-bit address of **0013A20012345678** needs to have its registration information removed from a trust center so that the device remains on the network, but can no longer securely join. After de-registration, a remote **NR0** command can be issued to remove the device from the network. This example only applies to a distributed trust center network—**EO** = **0**—a centralized trust center will automatically expire the entry after **KT** seconds.

The corresponding Registration Status - 0xA4 response with a matching Frame ID will indicate whether the de-registration succeeded.

```
7E 00 0D 24 D5 00 13 A2 00 12 34 56 78 FF FE 00 40
```

Frame type	Frame ID	64-bit reg address	Reserved	Options	Key/Install Code
0x24	0xD5	0x0013A200 12345678	0xFFFE	0x00	(omitted)
Registration	*Matches response*	*device to de-register*	*N/A*	*N/A*	*Remove entry*

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 Zigbee 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: User Data Relay Output - 0xAD

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 User Data Relay Output - 0xAD 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 1/10 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"*

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 Discovery). 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*

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 **0x43** = Open Join Window **0x44** = Closed Join Window **0x45** = Network Key Rotation initiated **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 Zigbee

0x00 = Hardware reset or power up

0x01 = Watchdog timer reset

0x02 = Joined network (routers and end devices)

0x03 = Disassociated

0x06 = Coordinator started

0x07 = Network security key was updated

0x0D = Voltage supply limit exceeded—see **Over-voltage detection** in the *XBee 3 RF Module Hardware Reference Manual*.

0x11 = Modem configuration changed while join in progress

0x3B = XBee 3 - Secure session successfully established

0x3C = XBee 3 - Secure session ended

0x3D = XBee 3 - Secure session authentication failed

0x3E = XBee 3 - Coordinator detected a PAN ID conflict but because **CR** = **0**, no action will be taken.

0x3F = Coordinator changed PAN ID due to a conflict

0x32 = XBee 3 - BLE Connect

0x33 = XBee 3 - BLE Disconnect

0x34 = XBee 3 - No Secure Session Connection

0x40 = Router PAN ID was changed by coordinator due to a conflict

0x42 = Network Watchdog timeout expired three times

0x80 - **0xFF** = Stack error

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 Zigbee

0x00 = Success

0x01 = MAC ACK Failure

0x02 = CCA Failure

0x15 = Invalid destination endpoint

0x21 = Network ACK Failure

0x22 = Not Joined to Network

0x23 = Self-addressed

0x24 = Address Not Found

0x25 = Route Not Found
0x26 = Broadcast source failed to hear a neighbor relay the message
0x2B = Invalid binding table index
0x2C = Resource error lack of free buffers, timers, etc.
0x2D = Attempted broadcast with APS transmission
0x2E = Attempted unicast with APS transmission, but **EE** = **0**
0x32 = Resource error lack of free buffers, timers, etc.
0x34 = XBee 3 - No Secure Session Connection
0x35 = Encryption Failure
0x74 = Data payload too large
0x75 = Indirect message unrequested

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.

Transmit Status - 0x89

Request frames:

- TX Request: 64-bit address frame - 0x00
- TX Request: 16-bit address - 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 Zigbee

This frame type is only used for indicating errors in sending a User Data Relay request

0x7C = Invalid interface. See User Data Relay Input - 0x2D.

0x7D = Interface not accepting frames. See User Data Relay Input - 0x2D.

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 TX Request: 64-bit address frame - 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*

Receive Packet - 0x90

Request frames:

- Transmit Request - 0x10
- Explicit Addressing Command Request - 0x11

Description

This frame type is emitted when a device configured with standard API output—AO (API Options) = **0**—receives an RF data packet.

Typically this frame is emitted as a result of a device on the network sending serial data using the Transmit Request - 0x10 or Explicit Addressing Command Request - 0x11 addressed either as a broadcast or unicast transmission.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Receive Packet - **0x90**
4	64-bit	**64-bit source address**	The sender's 64-bit address.
12	16-bit	**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**: Reserved ■ **Bit 3**: Reserved ■ **Bit 4**: Packet was sent across a secure session [0x10] ■ **Bit 5**: Packet encrypted with Zigbee APS security [**0x20**] ■ **Bit 6**: 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 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**: Reserved ■ **Bit 4**: Packet was sent across a secure session [**0x10**] ■ **Bit 5**: Packet encrypted with Zigbee APS security [**0x20**] ■ **Bit 6**: 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 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 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*

Node Identification Indicator - 0x95

Description

This frame type is emitted when a node identification broadcast is received. The node identification indicator contains information about the identifying device, such as address, identifier string (**NI**), and other relevant data.

A node identifies itself to the network under these conditions:

- The commissioning button is pressed once.
- A **CB 1** command is issued.
- A device with JN (Join Notification) enabled successfully associates with a Zigbee network.
- A device that is associated with a Zigbee network that has JN (Join Notification) enabled is power cycled.

See ND (Network Discovery) for information on the payload formatting.

See NO (Network Discovery Options) for configuration options that modify the output of this frame.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Node Identification Indicator - **0x95**
4	64-bit	**64-bit source address**	The sender's 64-bit address.
12	16-bit	**16-bit source address**	The sender's 16-bit network address.

Offset	Size	Frame Field	Description
14	8-bit	**Options**	Bit field of options that apply to the received message: ■ Bit 0: Reserved ■ **Bit 1**: Packet was sent as a broadcast [**0x02**] ■ **Bit 2**: 802.15.4 only - Packet was broadcast across all PANs [**0x04**] ■ Bit 4: Reserved ■ Bit 5: Reserved ■ **Bit 6, 7**: DigiMesh delivery method • b'00 = <invalid option> • b'01 = Point-multipoint [**0x40**] • b'10 = Directed Broadcast [**0x80**] • b'11 = DigiMesh [**0xC0**] **Note** Option values may be combined.
15	16-bit	**16-bit remote address**	The 16-bit network address of the device that sent the Node Identification.
17	64-bit	**64-bit remote address**	The 64-bit address of the device that sent the Node Identification.
25	variable (2-byte minimum)	**Node identification string**	Node identification string on the remote device set by NI (Node Identifier). The identification string is terminated with a NULL byte (0x00).
27+NI	16-bit	**Zigbee 16-bit parent address**	Indicates the 16-bit address of the remote's parent or **0xFFFE** if the remote has no parent. Equivalent to MP (16-bit Parent Network Address).
29+NI	8-bit	**Network device type**	What type of network device the remote identifies as: 0 = Coordinator 1 = Router 2 = End Device
30+NI	8-bit	**Source event**	The event that caused the node identification broadcast to be sent. 0 = Reserved 1 = Frame sent by node identification pushbutton event—see D0 (DIO0/AD0/Commissioning Button Configuration). 2 = Frame sent after joining a Zigbee network—see JN (Join Notification). 3 = Frame sent after a power cycle event occurred while associated with a Zigbee network—see JN (Join Notification).
31+NI	16-bit	**Digi Profile ID**	The Digi application Profile ID—**0xC105**.

Offset	Size	Frame Field	Description
33+NI	16-bit	**Digi Manufacturer ID**	The Digi Manufacturer ID—**0x101E**.
35+NI	32-bit	**Device type identifier (optional)**	The user-defined device type on the remote device set by DD (Device Type Identifier). Only included if the receiving device has the appropriate NO (Network Discovery Options) bit set.
EOF-1	8-bit	**RSSI (optional)**	The RSSI of the last hop that relayed the message. Only included if the receiving device has the appropriate NO (Network Discovery Options) bit set.
EOF	8-bit	Checksum	0xFF minus the 8-bit sum of bytes from offset 3 to this byte—between length and checksum.

Examples

Each example is written without escapes (**AP** = **1**) and all bytes are represented in hex format. For brevity, the start delimiter, length, and checksum fields have been excluded.

Identify remote device

A technician is replacing a DigiMesh device in the field and needs to have the its entry removed from a cloud server's database. The technician pushes the commissioning button on the old device once to send an identification broadcast. The server can use the broadcast to identify which device is being replaced and perform the necessary action.

When the node identification broadcast is sent, every device that receives the message will flash the association LED and emit the following information frame:

```
7E 00 27 95 00 13 A2 00 12 34 56 78 FF FE C2 FF FE 00 13 A2 00 12 34 56 78 4C
48 37 35 00 FF FE 01 01 C1 05 10 1E 00 14 00 08 0D
```

Frame type	64-bit source	16-bit source	Options	16-bit remote	64-bit remote	NI String	Parent	Device type	Event	Profile ID	MFG ID
0x95	0x0013A200 12345678	0xFFFE	0xC2	0xFFFE	0x0013A200 12345678	0x4C483735 00	0xFFFE	0x01	0x01	0xC105	0x101E
Identification		*Unknown*	*DigiMesh broadcast*	*Unknown*		*"LH75" + null*	*Unknown*	*Router*	*Button press*	*Digi*	*Digi*

Identify joining device

A Zigbee end device has join notification enabled by setting **JN** to **1**. When the joining device successfully associates with a Zigbee network, it will broadcast a node identification message.

The network has a variety of devices that are assigned identifier strings after association; a unique **DD** value is set to identify this type of device. The gateway that manages the network has the **NO** command set to **1** to display this information.

When the node identification broadcast is sent, every device that receives the message will flash the association LED and emit the following information frame:

```
7E 00 24 95 00 13 A2 00 87 65 43 21 77 92 02 77 92 00 13 A2 00 87 65 43 21 20
00 45 A3 02 02 C1 05 10 1E 00 12 00 27 13
```

	Frame type	64-bit source	16-bit source	Options	16-bit remote	64-bit remote	NI String	Parent	Device type	Event	Profile ID	DD value
0x95	0x0013A20087654321	0x7792	0x02	0x7792	0x0013A20012345678	0x2000	0x45A3	0x02	0x02	0xC105	0x101E	0x00120027
Identification			*Broadcast*			*No NI string set*		*End device*	*Joined*	*Digi*	*Digi*	*Zigbee + User-defined*

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 (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

This frame type can be used to troubleshoot Zigbee network association. To enable verbose join information, use DC (Joining Device Controls).

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.

Status code	Description	Status data	Size	Description
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
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 DC (Joining Device Controls) 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
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	

Zigbee Verbose Join status codes

The following table describes the various Verbose Join trace messages in Status Code order. The Transparent mode string column shows the string which appears if you run Verbose Join in Command mode. The Description column gives a more detailed explanation of each particular message. When a message accompanies Status Data, the Status Data column shows how to parse the hexadecimal string into fields. The **Size** column shows the number of bytes per field.

Status code	Transparent mode string	Description	Status data	Size	Description
0x00	Rejoin	A join attempt is being started.	rejoinState	8-bit	The rejoinState is a count of join attempts.

Status code	Transparent mode string	Description	Status data	Size	Description
0x01	Stack Status	Shows status and state.	Status	8-bit	**0x00** - no network **0x01** - joining **0x02** - joined **0x03** - joined (no parent) **0x04** - leaving
			NetworkState	8-bit	**0x90** - Network is up and ready to receive/transmit. **0x91** - Network is down and cannot receive/transmit. **0x94** - Join attempt failed. **0x96** - A node's attempt to re-establish contact with the network after moving failed. **0x98** - A join attempt as a router failed due to a Zigbee 2006 versus Zigbee PRO 2007 incompatibility. Try to join as an end device. **0x99** - The network ID has changed. **0x9A** - The PAN ID has changed. **0x9B** - The channel has changed. **0xAB** - No beacons were received in response to a beacon request. **0xAC** - Received key in the clear. **0xAD** - No network key received. **0xAE** - No link key received. **0xAF** - Preconfigured key required—Settings for **KY** may not match.

Status code	Transparent mode string	Description	Status data	Size	Description
0x02	Joining	An association request is being made.	radioChannel	8-bit	Channel number ranging from 11 to 26 (0x0B to 0x1A)
			radioTxPower	8-bit	Low level signed byte value for transmit power, values range from 0xC9 to 0x05 inclusive
			panid	16-bit	16-bit PAN Identifier for the network
			extendedPanId	64-bit	64-bit extended PAN Identifier for network
0x03	Joined	Coordinator “Formed”, Router/End Device “Joined” and whether the network formed or joined is a "secure network" or an "unsecured network."			
0x04	Beacon Response	Data received from a neighboring node in response to a beacon request	ZS[stackProfile]	8-bit	See ZS (Zigbee Stack Profile).
			extendedPanId	64-bit	64-bit Extended PAN Identifier for network
			allowingJoin	8-bit	**0x00** - not permitting joins to its network **0x01** - permitting joins to its network
			radioChannel	8-bit	Channel number ranging from 11 to 26—0x0B to 0x1A
			panid	16-bit	16-bit PAN Identifier for network
			rssi	8-bit	Maximum relative signal strength indicator value measured in units of dBm (applies to last hop only)
			lqi	8-bit	Link quality indicator
0x05	Reject **ZS**	Not an association candidate because **ZS** does not match that given in the beacon response.			

Status code	Transparent mode string	Description	Status data	Size	Description
0x06	Reject **ID**	Not an association candidate because configured pan **ID** does not match that given in the beacon response.			
0x07	Reject **NJ**	Not an association candidate because it is not allowing joins.			
0x08	panID Match	**JV**/**NW** with search option (DO80) has found a matching network.	panId	16-bit	16-bit PAN Identifier for network
0x09	Reject LQIRSSI	JV/NWwith search option (DO80) candidate rejected because this beacon response is weaker than an earlier beacon response.			
0x0A	Beacon Saved	This beacon response is a suitable candidate for an association request.	radioChannel	8-bit	Channel number ranging from 11 to 26 (0x0B to 0x1A)
			radioTxPower	8-bit	Low level signed byte value for transmit power, values range from 0xC9 to 0x05 inclusive
			panid	16-bit	16-bit PAN Identifier for network
			extendedPanId	64-bit	64-bit Extended PAN Identifier for network
0x0B	**AI**	**AI** value has changed.	AIStatusCode	8-bit	See a description of AI (Association Indication)
0x0C	Permit Join	**NJ** setting (Permit Join Duration) has changed	value	8-bit	See a description of the NJ (Node Join Time) command.
0x0D	Scanning	Active scanning has begun.	ChannelMask	32-bit	A 32-bit value driven by the **SC** setting where bit positions 11 through 26 show which channels are enabled for the upcoming Active Scan. See a description of SC (Scan Channels).
0x0E	Scan Error	An error occurred during active scan.	StatusCode	8-bit	

Status code	Transparent mode string	Description	Status data	Size	Description
0x0F	Join Request	High level request for a form/join.			
0x10	Reject LQI	Reject because LQI is worse than an already saved beacon	lqi	8-bit	Link quality indicator
0x11	Reject RSSI	Rejected because RSSI is worse than an already saved beacon	rssi	8-bit	Relative signal strength indicator
0x12	Rejected (cmdLast)	Rejected because it matches the last associated network.			
0x13	Rejected (cmdSave)	Rejected because it matches an already saved beacon response.			
0x14	Reject strength	During first/best phase, response is weaker than an already saved beacon response.			
0x16	Reset for DC80	With DC80 enabled, reset if no joinable beacon responses are received within 60s of joining.			
0x18	ScanCh	Scanning on Channel	radioChannel	8-bit	Channel number ranging from 11 to 26 (0x0B to 0x1A)
0x19	Scan Mode	Shows phase of Ordered Association.	mode	8-bit	**0x00**: First/best candidate **0x01**: Ordered association by extpanid, then by channel
0x1A	Scan Init	Starting a scan	channel	8-bit	Channel being scanned
			TxPower	8-bit	Low level radio transmit power setting
0x1D	Energy Scan - channel mask	Starting energy scan	**SC** mask	32-bit	Scan channel mask
0x1E	Energy Scan - energies	Channel Energies observed	Energies	128-bit	Energy Levels per channel in **SC**

Status code	Transparent mode string	Description	Status data	Size	Description
0x1F	PanIdScan - radio channel	Pan Id Scan starting on channel	channel	8-bit	Radio Channel
0x20	FormNetwork - parameters	Forming a network	radioChannel	8-bit	Channel number ranging from 11 to 26
			radioTxPower	8-bit	Low level radio transmit power setting
			panid	16-bit	16-bit PAN identifier for network
			extendedpanid	64-bit	64-bit Extended PAN identifier for network
0x21	Discovering **KE** Endpoint	Looking for Key Establishment Endpoint			
0x22	**KE** Endpoint	Found Key Establishment Endpoint	Endpoint	8-bit	Endpoint number
0x23	Key exchange timeout	The key exchange process timed out.			
0x24	Key established	The key has been established.			
0x25	Key verified	The key has been verified.			
0x26	Need new key	A key exchange is required to join the network.			
0x27	Key join done	The key exchange process has completed successfully.			
0x28	Ch verify fail	Channel verification failed during the join process.			
0x29	LK update fail	The link key update failed during the join process.			
0x2A	Key verify timeout	The link key update timed out during the join process.			

Status code	Transparent mode string	Description	Status data	Size	Description
0x2B	Unknown key fail	An unknown error occurred in the key exchange process.			
0x2C	Request from:	Coordinator only. A request to join was received.	EUI64	64-bit	The EUI64 of the radio requesting to join.
0x2D	EUI64 in key table	Coordinator only. A centralized trust center has found an entry in the transient key table for a device requesting to join the network.			
0x2E	"Joining allowed:" or "Joining not allowed:"	Coordinator only. Indicate whether or not joining is currently allowed.	Joining allowed	8-bit	1 if joining is allowed, else 0.

Route Record Indicator - 0xA1

Description

This frame type contains the routing information for a remote device on the network. This route information should be stored in external memory and used in a Create Source Route - 0x21 frame to provide a return route for subsequent data transmissions; this eliminates the need to perform a route discovery.

This frame type is emitted when a network concentrator receives a route record from a remote device. The type of concentrator determines how often this frame type is emitted: a high RAM concentrator (the default) will emit this frame type when a unicast data transmission is received for the first time. If a previously established route fails, a new 0xA1 Route Record Indicator will be generated. A low RAM concentrator will emit this frame for every received transmission. Concentrator type is determined by DO (Miscellaneous Device Options) bit 6.

Format

The following table provides the contents of the frame. For details on frame structure, see API frame format.

Offset	Size	Frame Field	Description
0	8-bit	Start Delimiter	Indicates the start of an API frame.
1	16-bit	Length	Number of bytes between the length and checksum.
3	8-bit	**Frame type**	Route Record Indicator - **0xA1**
5	64-bit	**64-bit source address**	The 64-bit IEEE address of the device that initiated the route record.
13	16-bit	**16-bit source address**	The 16-bit network address of the device that initiated the route record.
15	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**]
16	8-bit	**Number of addresses**	The number of addresses in the source route—excluding source and destination.
17-n	16-bit variable	**Address**	The 16-bit network address(es) of the devices along the source route, excluding the source and destination. The addresses are in order from destination to source and match the order to be entered into the Create Source Route - 0x21 frame.

Offset	Size	Frame Field	Description
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.

4-hop route

A remote device sends a unicast transmission to a concentrator that is 4-hops away. The concentrator emits a route record that can be stored for use in a subsequent Create Source Route - 0x21 frame prior to sending data back to the remote.

The route looks like this:

Destination (concentrator) <> **Router A** <> **Router B** <> **Router C** <> **Source (remote)**

```
7E 00 13 A1 00 13 A2 00 12 34 56 78 DD DD 01 03 CC CC BB BB AA AA 75
```

Frame type	64-bit source	16-bit source	Options	Num of addresses	Address 1	Address 2	Address 3
0xA1	0x0013A200 12345678	0xDDDD	0x01	0x03	0xCCCC	0xBBBB	0xAAAA
Route	*Source IEEE address*	*Source NWK address*		*3*	*Neighbor of source*	*Intermediate hop*	*Neighbor of destination*

Registration Status - 0xA4

Request frame: Register Joining Device - 0x24

Description

This frame type is emitted in response to registering a device to a trust center using the Register Joining Device - 0x24 frame and indicates whether the registration attempt succeeded or not.

This frame is only emitted if the 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**	Register Device Status - **0xA4**
4	8-bit	**Frame ID**	Identifies the data frame for the host to correlate with a prior request.
5	8-bit	**Registration status**	Status code for the registration request: **0x00** = Success **0x01** = Key too long **0x18** = Transient key table is full **0xB1** = Address not found in the key table **0xB2** = Key is invalid (00 and FF are reserved) **0xB3** = Invalid address **0xB4** = Key table is full **0xBD** = Security data is invalid (Install code CRC fails)
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.

Successful registration

A device is registered with a trust center using a Register Joining Device - 0x24 frame.

The corresponding 0xA4 Registration Status response with a matching Frame ID is emitted as a response:

```
7E 00 03 A4 5D 00 FE
```

Frame type	Frame ID	Status
0xA4	0x5D	0x00
Response	*Matches request*	*Success*

Many-to-One Route Request Indicator - 0xA3

Description

This frame type is emitted on devices that receive a many-to-one route request from a network concentrator. Typically, a device that emits this frame type should send a unicast message to the sender so a route record can be generated.

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**	Many-to-One Route Request Indicator - **0xA3**
4	64-bit	**64-bit source address**	The 64-bit IEEE address of the device that sent the many-to-one route request.
12	16-bit	**16-bit source address**	The 16-bit network address of the device that sent the many-to-one route request.
14	8-bit	**Receive options (reserved)**	Options are not available yet. This bit field is reserved for future functionality. This field returns 0.
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.

Many-to-one route request

A gateway is configured for many-to-one routing by setting AR (Aggregate Routing Notification) to send a many-to-one route request every 10 minutes—**AR** = **0x3C**.

Whenever an aggregator broadcast is sent, the following frame is emitted on all devices:

```
7E 00 0C A3 00 13 A2 00 87 65 43 21 00 00 00 57
```

Frame type	64-bit source	16-bit source	Reserved
0xA3	0x0013A200 87654321	0x0000	0x00
MTORR		*NWK address*	

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.
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

Offset	Size	Frame Field	Description
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.
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 Zigbee 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 Zigbee 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	The Zigbee stack version used by the application. This field is informational for the server and is not used during the upgrade process. **Note** For XBee 3 Zigbee firmwares, the value of this field is **2** indicating Zigbee Pro—see ZCL Spec §11.4.2.8 for a full list of values. The actual Zigbee stack profile used by the device may differ depending on the value of the **ZS** command.
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.

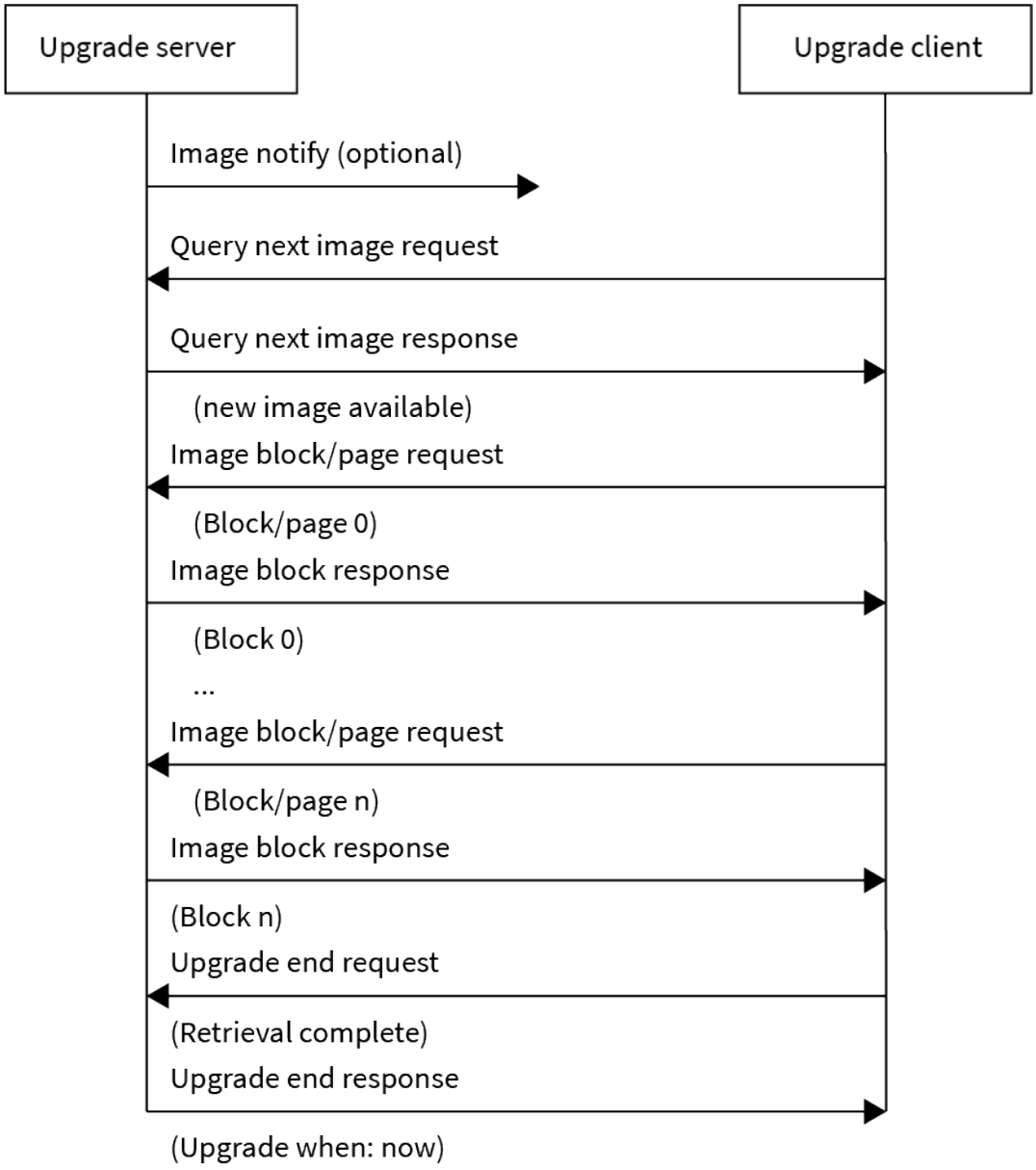
Offset	Length	Field name	Description
0	2	Sub-element tag	The tag for the sub-element, in little-endian format. This is usually 0x0000 for 'upgrade image'—this is the case for both firmware upgrades and file system upgrades.
2	4	Sub-element length	The length of the sub-element data (n) in little-endian 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 100A. 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 Zigbee 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 100A. 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 Zigbee 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 100A. 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 Zigbee 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.
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 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 Zigbee 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 1 to one—for example, **SO** = 0x01.

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** (router)
      - **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.

Version 1009 and prior: Only the GBL file is sent over the air

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?

Version 1009 and prior: Delayed ACK for some packets

The Query Next Image Response and the final Image Block Response both cause the client to perform a long operation—erasing/verifying OTA update data in the storage slot. On these versions, the network ACK for the transmission is not sent until after this operation completes. This means that the server will time out waiting for an ACK, and the transmission will appear to fail even though it was in fact received by the client.

On Zigbee, this error can be worked around by enabling the extended APS timeout. Set bit six of the Transmit Options field (0x40) in the transmit API frame, at least when sending a Query Next Image Response or the final Image Block Response.

Version 1008 and prior: Recovering from failed client transmissions

On XBee 3 Zigbee versions 1008 and previous, the OTA client will not attempt to resend a request to the server if it fails due to network conditions. This is known to occur in large networks when the client attempts to send the first Image Block Request or the Upgrade End Request, but could happen at any time during the download due to a failed transmission.

When this occurs, it will appear to the server as though the client stopped sending requests. The server can recover the update from this state using the following method:

1. When sending messages to the client, the OTA server should check the TX status of the transmission to ensure it was successfully delivered to the client.
2. When the server waits for the client's response, it should do so with a timeout.
    - How long this timeout should be can vary based on network settings, but something like 20 seconds should cover most cases.
    - Keep in mind when determining the timeout that the client can take a long time (6-8 seconds) to process some OTA commands, such as the Query Next Image Response or the final Image Block Response.
3. If the server does not receive the request from the client, the server should proceed and send the next response, as though it had received the expected request from the client.
    - For this process, note that firmware versions 1008 and prior will always specify 64 for the **maximum data size** field of an Image Block Request if encryption is disabled, and 44 if it is enabled. Use these as the maximum sizes when constructing image blocks.
    - For example, if the last packet sent before the timeout was a Query Next Image Response, the server should send an Image Block Request with the first image block, for example 64 bytes—or 44 on an encrypted network—starting at offset 0.
    - If the last packet sent before the timeout was an Image Block Response, the server should send another Image Block Response starting immediately after the end of the last block sent and of the same size.
    - If the last packet sent before the timeout was an Image Block Response containing the final bytes of the image, the server should send an Upgrade End Response.
4. The client will accept the generated response and continue the update. Failures like this should not happen consistently; If the server times out and has to do this for more than about

three packets in a row without hearing from the client, it should assume communication with the client is lost and the update has failed.

Note This method should only be used when updating from a version 1008 or prior. On newer versions the client will resend failed requests, and sending a response unprompted like this could cause an error in the download.

Version 1007 and prior: OTA commands cannot be sent with fragmentation

An OTA update with any of these versions as the client will fail if an OTA command is sent with fragmentation. This is why Image Block Requests sent by these versions set the maximum data size to 64 or 44 bytes. However, on networks with source routing enabled—this includes any network that is using encryption—this means that if a message is sent with a route record included then the payload size must be decreased even more to ensure fragmentation is not used. This can be handled one of two ways by the server:

1. Set **AR** to **0xFF** on the server for the duration of the update. This will ensure that the OTA server does not send route records to the client, and is the recommended method when updating devices from version 1007 or earlier.
2. Alternatively, if the number of relays between the server and the client is known, the image block size can be reduced to accommodate the reduced payload size.
    - The payload should be reduced by one byte plus two byes for each relay node—including the client.
    - The payload should then be further reduced to a multiple of four.
    - Be aware that the route could change during the update due to changing network conditions, so it would be wise to include some additional overhead.

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 100A and newer, the entire OTA file is sent to the client during an OTA Upgrade. However, for versions older than 100A, 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-100A)	Value when sending with header (100A 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 Zigbee 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

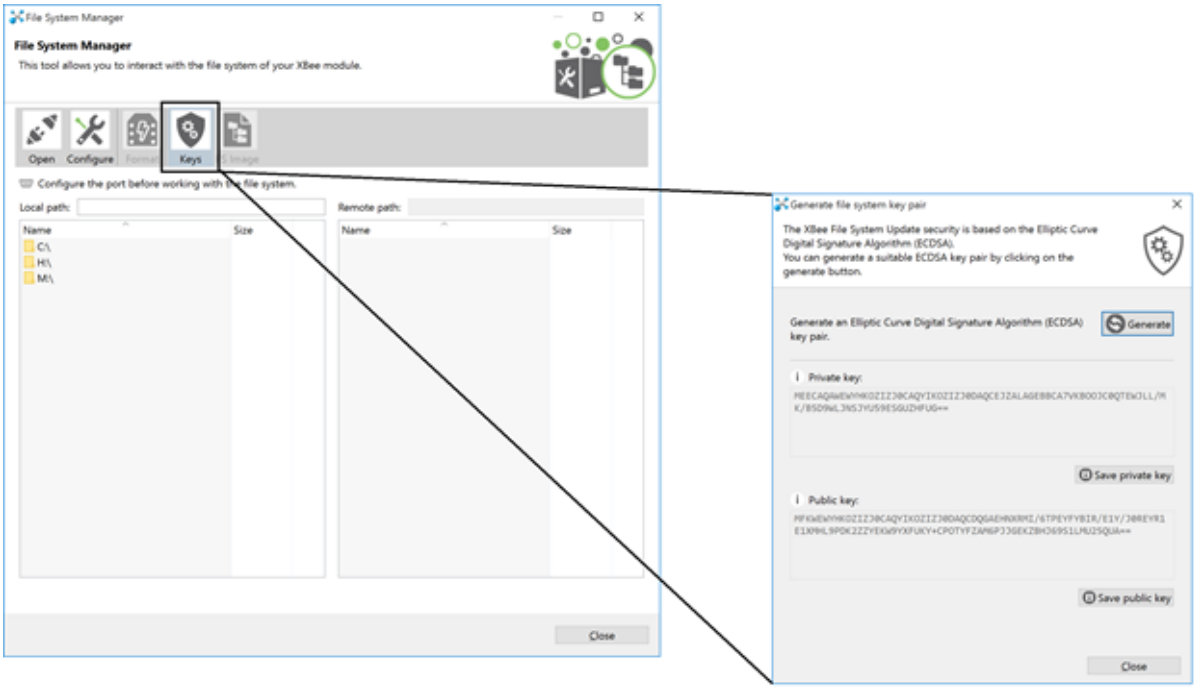
Use the following steps to perform a file system update OTA using XCTU:

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:

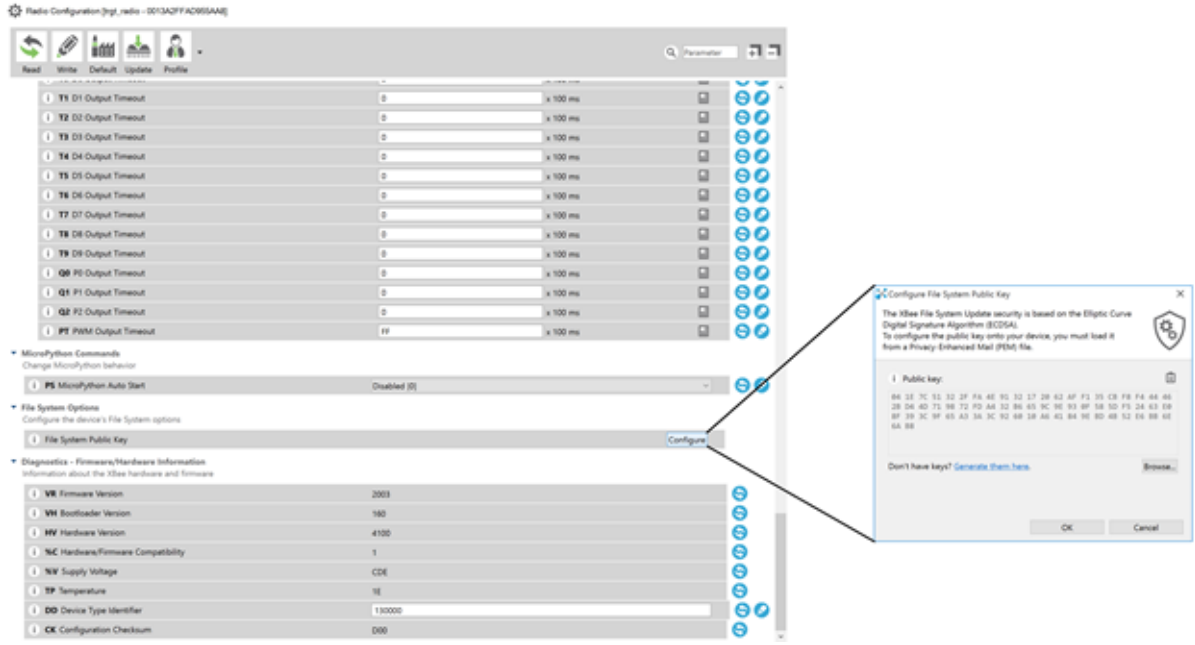
1. Open the **File System Manager** dialog box.
2. Click **Keys** as shown below.



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**.



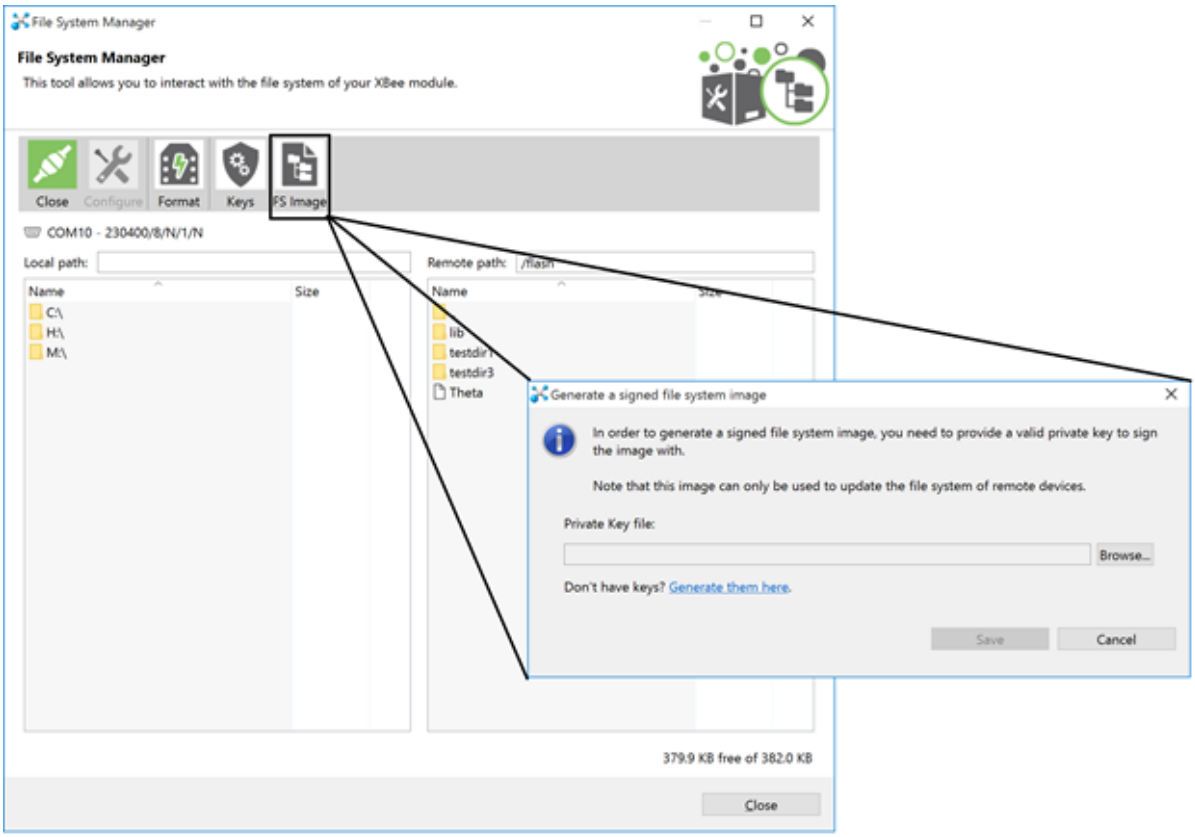
3. In the **Configure File System Public Key** dialog box, click **Browse** and choose the .pem file that you saved the public key into. Once this is done, the HEX value of the public key is visible under the **Public key** section on the dialog box as shown.
4. Click **OK** to ensure that the key gets 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:

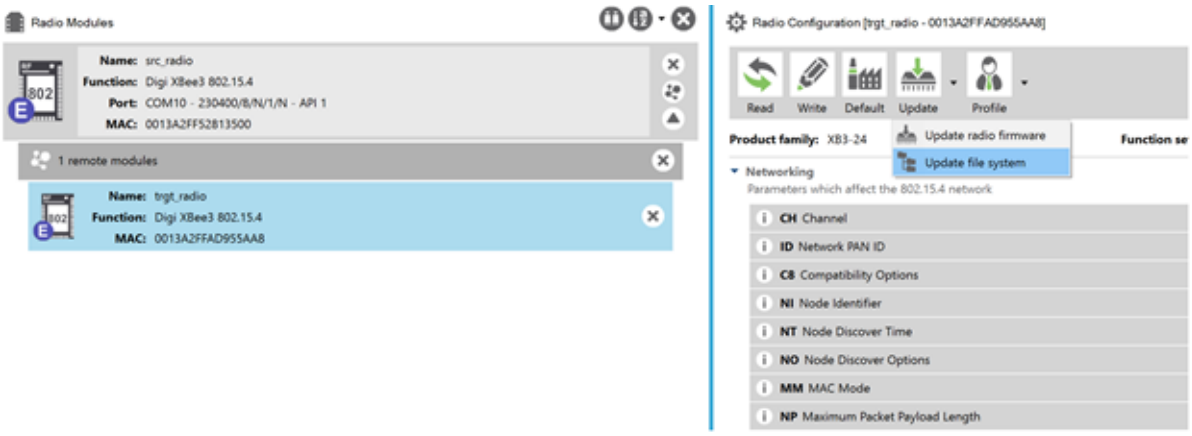
1. Open the **File System Manager** dialog box.
2. Open a connection on the device that you want to generate the OTA file system image from.
3. Click **FS Image**.
4. In the **Generate a signed file system image** window that displays, click **Browse** and choose the .pem file that the private key was 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.



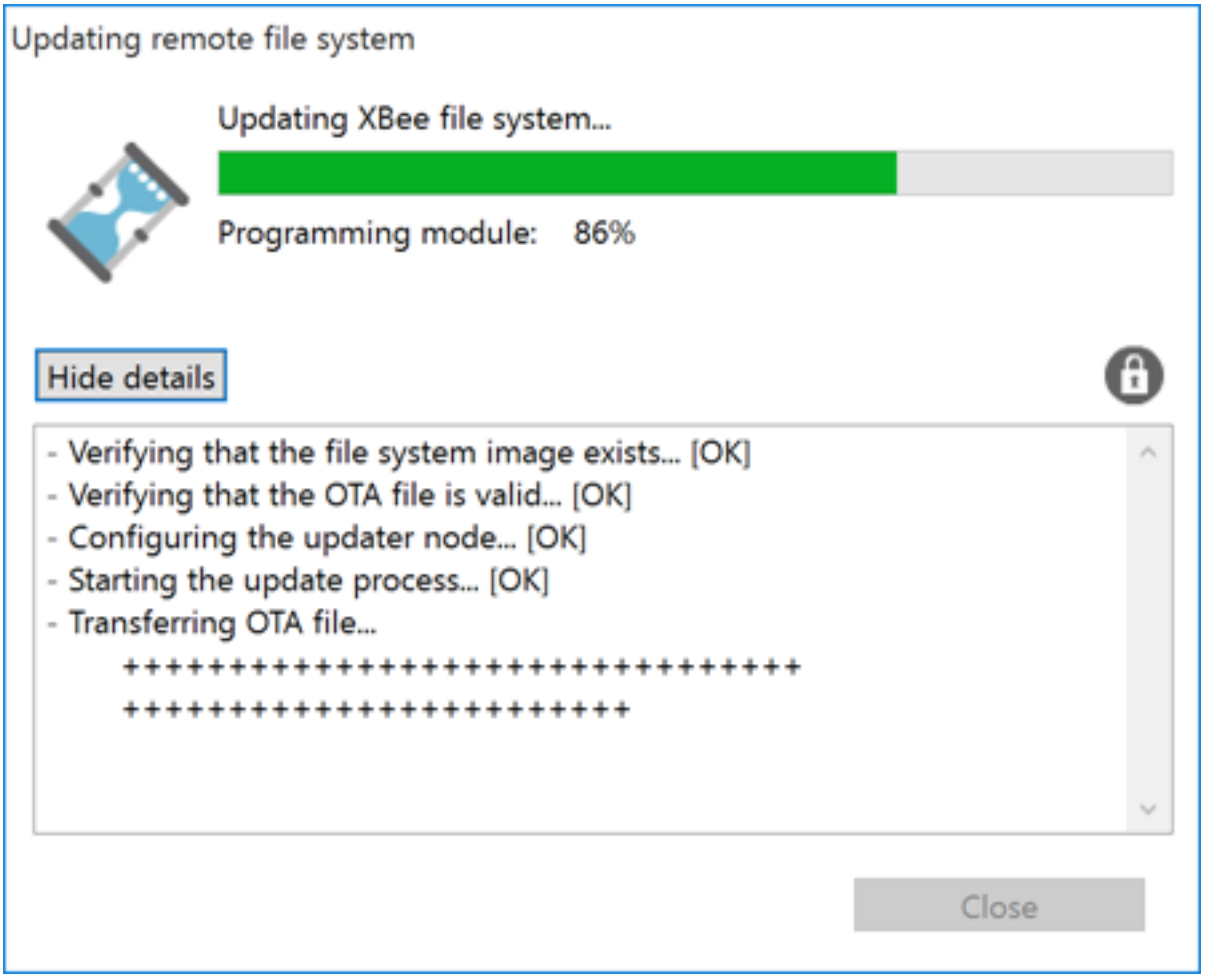
You will be prompted with a **File system image successfully saved** dialog box if the file was successfully generated.

Perform the OTA file system update

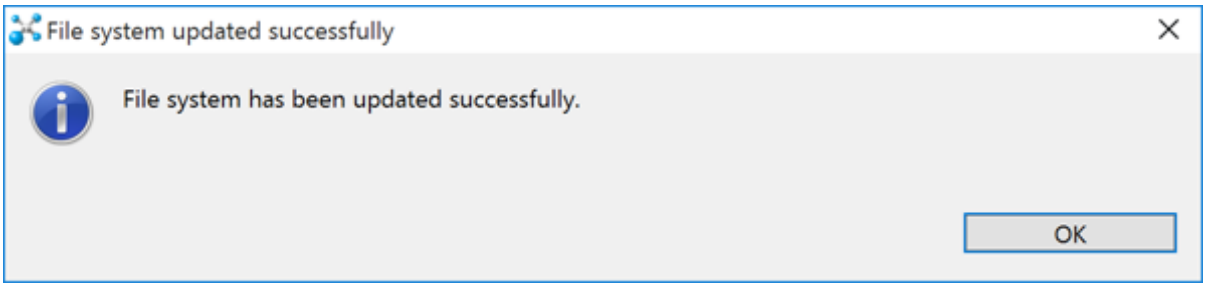
1. To add the target device, click **Discover radios in the same network** from the source device.
2. Enter Configuration mode on the remote device.
3. Click the down arrow next to the **Update** button and choose **Update File System**.



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:

```
049550aa55b6f55d994dd815d171575180d514ec1f6a1551a2c4b80f77108a33a380074740148b5
ca74c7802fc4d82904b399862a11d976e78fb546206d241c73b
```

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/file system upgrade process for Zigbee 3.0. 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.