



Digi XBee3[®] DigiMesh 2.4

RF Module

User Guide

Revision history—90002277

Revision	Date	Description
A	April 2018	Initial release.
B	September 2018	Added features for S2C parity.

Trademarks and copyright

Digi, Digi International, and the Digi logo are trademarks or registered trademarks in the United States and other countries worldwide. All other trademarks mentioned in this document are the property of their respective owners.

© 2018 Digi International Inc. All rights reserved.

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@digicom.com

Include the document title and part number (Digi XBee3® DigiMesh 2.4 RF Module User Guide, 90002277 B) in the subject line of your email.

Contents

About the XBee3 DigiMesh RF Module

Applicable firmware and hardware	12
Change the firmware protocol	12
Regulatory information	12

Get started

Verify kit contents	14
Assemble the hardware	14
Plug in the XBee3 DigiMesh RF Module	15
Unplug an XBee3 DigiMesh RF Module	16
Configure the device using XCTU	16
Configure remote devices	16
Configure the devices for a range test	17
Perform a range test	18

Configure the XBee3 DigiMesh RF Module

Software libraries	22
Over-the-air (OTA) firmware update	22
Custom defaults	22
Set custom defaults	22
Restore factory defaults	22
Limitations	22
XBee Network Assistant	23

Modes

Transparent operating mode	25
API operating mode	25
Command mode	25
Enter Command mode	25
Troubleshooting	26
Send AT commands	26
Response to AT commands	26
Apply command changes	27
Make command changes permanent	27
Exit Command mode	27
Idle mode	27

Transmit mode	27
Receive mode	27

Serial communication

Serial interface	29
Serial receive buffer	29
Serial transmit buffer	29
UART data flow	29
Serial data	30
Flow control	30
Clear-to-send (CTS) flow control	31
RTS flow control	31

SPI operation

SPI communications	33
Full duplex operation	34
Low power operation	34
Select the SPI port	35
Force UART operation	36

I/O support

Digital I/O support	38
Analog I/O support	38
Monitor I/O lines	39
I/O sample data format	40
API frame support	41
On-demand sampling	41
Example: Command mode	41
Example: Local AT command in API mode	42
Example: Remote AT command in API mode	42
Periodic I/O sampling	43
Source	43
Destination	44
Digital I/O change detection	44
I/O line passing	44
Digital line passing	45
Example: Digital line passing	45
Analog line passing	45
Example: Analog line passing	46
Output sample data	46
Output control	46
I/O behavior during sleep	46
Digital I/O lines	47
Analog and PWM I/O Lines	47

Networking

Network identifiers	49
Operating channels	49
Delivery methods	49

Point-to-multipoint	49
DigiMesh networking	50
Broadcast addressing	50
Unicast addressing	51
Route discovery	51
Routing	52
Routers	52
Repeater/directed broadcast	52
MAC layer	52
Encryption	53
Maximum payload	53

Network commissioning and diagnostics

Local configuration	55
Remote configuration	55
Send a remote command	55
Apply changes on remote devices	55
Remote command response	55
Build aggregate routes	56
DigiMesh routing examples	56
Replace nodes	57
Test links between adjacent devices	57
Trace route option	59
NACK messages	59
RSSI indicators	60
Associate LED	60
The Commissioning Pushbutton	60
Definitions	61
Use the Commissioning Pushbutton	61
Node discovery	62
Discover all the devices on a network	62
Directed node discovery	62
Destination Node	63
Discover devices within RF range	63

Sleep support

Sleep modes	65
Pin Sleep mode (SM = 1)	65
Cyclic Sleep mode (SM = 4)	65
Cyclic Sleep with Pin Wake-up mode (SM = 5)	66
Sleep parameters	66
Sleep current	66
Sleep pins	66
Sleep conditions	67

AT commands

Networking commands	69
CH (Operating Channel)	69
ID (Network ID)	69
CE (Routing / Messaging Mode)	69

C8 (Compatibility Options)	70
NI (Network Identifier)	70
ND (Network Discover)	72
DN (Discover Node)	72
FN (Find Neighbors)	73
NT (Network Discovery Back-off)	74
NO (Network Discovery Options)	74
NP (Maximum Packet Payload Bytes)	74
DigiMesh Addressing commands	75
SH (Serial Number High)	75
SL (Serial Number Low)	75
DH (Destination Address High)	75
DL (Destination Address Low)	75
RR (Unicast Mac Retries)	76
MT (Broadcast Multi-Transmits)	76
TO (Transmit Options)	76
CI (Cluster ID)	77
DigiMesh configuration commands	77
MR (Mesh Unicast Retries)	77
BH (Broadcast Hops)	77
NH (Network Hops)	78
NN (Network Delay Slots)	78
DM (DigiMesh Options)	78
AG (Aggregator Support)	79
Diagnostic commands - addressing timeouts	79
%H (MAC Unicast One Hop Time)	79
%8 (MAC Broadcast One Hop Time)	79
N? (Network Discovery Timeout)	80
Security commands	80
EE (Encryption Enable)	80
KY (AES Encryption Key)	80
RF interfacing commands	81
PL (TX Power Level)	81
PP (Output Power in dBm)	81
CA (CCA Threshold)	81
DB (Last Packet RSSI)	82
MAC diagnostics commands	82
EA (MAC ACK Failure Count)	82
EC (CCA Failures)	82
BC (Bytes Transmitted)	83
GD (Good Packets Received)	83
TR (Transmission Failure Count)	83
UA (Unicasts Attempted Count)	84
ED (Energy Detect)	84
Sleep settings commands	84
SM command	84
SP (Sleep Time)	85
ST (Wake Time)	85
SN (Number of Sleep Periods)	85
WH (Wake Host Delay)	85
SO (Sleep Options)	86
UART interface commands	86
BD (Baud Rate)	86
NB (Parity)	87
SB (Stop Bits)	87

FT (Flow Control Threshold)	88
RO (Packetization Timeout)	88
AP (API Enable)	88
AO (API Options)	88
AZ (Extended API Options)	89
Command mode options	89
CC (Command Character)	89
CT (Command Mode Timeout)	90
GT (Guard Time)	90
CN (Exit Command mode)	90
UART pin configuration commands	90
D6 (DIO6/RTS Configuration)	90
D7 (DIO7/CTS Configuration)	91
P3 (DIO13/UART_DOUT)	91
P4 (DIO14/UART_DIN Configuration)	92
SPI interface commands	92
P5 (DIO15/SPI_MISO Configuration)	92
P6 (DIO16/SPI_MOSI Configuration)	93
P7 (DIO17/SPI_SSEL Configuration)	93
P8 (DIO18/SPI_CLK Configuration)	94
P9 (DIO19/SPI_ATTN Configuration)	94
I/O settings commands	95
D0 (DIO0/ADC0/Commissioning Configuration)	95
D1 (DIO1/ADC1/TH_SPI_ATTN Configuration)	95
D2 (DIO2/ADC2/TH_SPI_CLK Configuration)	96
D3 (DIO3/ADC3/TH_SPI_SSEL Configuration)	96
D4 (DIO4/TH_SPI_MOSI Configuration)	97
D5 (DIO5/Associate Configuration)	97
D8 (DIO8/DTR/SLP_Request Configuration)	98
D9 (DIO9/ON_SLEEP Configuration)	98
P0 (DIO10/RSSI/PWM0 Configuration)	99
P1 (DIO11/PWM1 Configuration)	99
P2 (DIO12/TH_SPI_MISO Configuration)	100
PR (Pull-up/Down Resistor Enable)	100
PD (Pull Up/Down Direction)	101
IO (Set Digital I/O Lines)	102
M0 (PWM0 Duty Cycle)	102
M1 (PWM1 Duty Cycle)	102
RP command	103
LT command	103
CB (Commissioning Button)	103
I/O sampling commands	104
IS (I/O Sample)	104
IR (Sample Rate)	104
IC (DIO Change Detect)	104
AV (Analog Voltage Reference)	105
IF (Sleep Sample Rate)	106
I/O line passing commands	106
IA (I/O Input Address)	106
IU (Send I/O Sample to Serial Port)	106
T0 (D0 Timeout)	107
T1 (D1 Output Timeout)	107
T2 (D2 Output Timeout)	107
T3 (D3 Output Timeout)	107
T4 (D4 Output Timeout)	108

T5 (D5 Output Timeout)	108
T6 (D6 Output Timeout)	108
T7 (D7 Output Timeout)	108
T8 (D8 Timeout)	108
T9 (D9 Timeout)	109
Q0 (P0 Timeout)	109
Q1 (P1 Timeout)	109
Q2 (P2 Timeout)	109
PT (PWM Output Timeout)	110
Diagnostics – Firmware/Hardware Information	110
VR (Firmware Version)	110
VL (Version Long)	110
VH (Bootloader Version)	110
HV (Hardware Version)	110
%C (Hardware/Software Compatibility)	111
%V (Supply Voltage)	111
TP (Temperature)	111
DD (Device Type Identifier)	111
CK (Configuration CRC)	112
FR (Software Reset)	112
Memory access commands	112
AC (Apply Changes)	112
WR (Write)	112
RE (Restore Defaults)	113
Custom default commands	113
%F (Set Custom Default)	113
!C (Clear Custom Defaults)	113
R1 (Restore Factory Defaults)	113

Operate in API mode

API mode overview	116
Use the AP command to set the operation mode	116
API frame format	116
API operation (AP parameter = 1)	116
API operation with escaped characters (AP parameter = 2)	117

Frame descriptions

AT Command Frame - 0x08	121
AT Command - Queue Parameter Value frame - 0x09	123
Transmit Request frame - 0x10	125
Explicit Addressing Command frame - 0x11	128
Remote AT Command Request frame - 0x17	131
AT Command Response frame - 0x88	133
Modem Status frame - 0x8A	135
Transmit Status frame - 0x8B	136
Route Information Packet frame - 0x8D	138
Aggregate Addressing Update frame - 0x8E	141
Receive Packet frame - 0x90	143
Explicit Rx Indicator frame - 0x91	145
I/O Data Sample Rx Indicator frame - 0x92	148
Node Identification Indicator frame - 0x95	150
Remote Command Response frame - 0x97	154

OTA firmware upgrade process for DigiMesh 2.4

OTA/OTB file	157
The OTA header	157
Hardware/software compatibility	158
Parse the image blocks	158
Storage	158
ZCL OTA messaging	158
ZCL message output	159
Image Notify	159
Create the Image Notify request	160
Query Next Image request	162
Query Next Image response	164
Image Block request	166
Image Block response	168
Upgrade End request	171
Upgrade End response	173
OTA error handling	175
Default response commands	175
Upgrade End Request error statuses	176

About the XBee3 DigiMesh RF Module

The XBee3 DigiMesh RF Module consists of DigiMesh 2.4 firmware loaded on the XBee3 hardware. This user guide covers the firmware. For information about XBee3 hardware, see the [XBee3 RF Module Hardware Reference Manual](#).

Digi XBee3 devices offer the flexibility to switch between multiple frequencies and wireless protocols as needed. These devices use the DigiMesh networking protocol using a globally deployable 2.4 GHz transceiver. This peer-to-peer mesh network offers users added network stability through self-healing, dense network operation, extending the operational life of battery dependent networks and provides an upgrade path to IEEE 802.15.4 or ZigBee mesh protocols, if desired.

Applicable firmware and hardware	12
Change the firmware protocol	12
Regulatory information	12

Applicable firmware and hardware

This manual supports the following firmware:

- 3001

It supports the following hardware:

- XBee3

Change the firmware protocol

You can switch the firmware loaded onto the XBee3 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


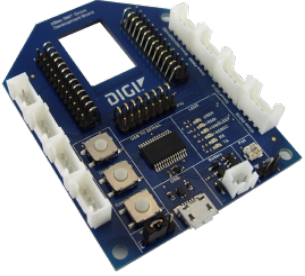



See the [Regulatory information](#) section of the [XBee3 RF Module Hardware Reference Manual](#) for the XBee3 hardware's regulatory and certification information.

Get started

Verify kit contents	14
Assemble the hardware	14
Configure the device using XCTU	16
Configure remote devices	16
Configure the devices for a range test	17
Perform a range test	18

Verify kit contents

The XBee3 DigiMesh RF Module development kit contains the following components:

Part	
XBee3 Zigbee SMT module (3)	
XBee Grove development board (3)	
Micro USB cable (3)	
Antenna - 2.4 GHz, half-wave dipole, 2.1 dBi, U.FL female, articulating (3)	
XBee stickers	

Assemble the hardware

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

- [Plug in the XBee3 DigiMesh RF Module](#)
- [Unplug an XBee3 DigiMesh RF Module](#)

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

Plug in the XBee3 DigiMesh RF Module

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

1. Plug one XBee3 DigiMesh RF Module into the XBee Grove Development Board. When you connect the development board to a PC for the first time, the PC automatically installs drivers, which may take a few minutes to complete.

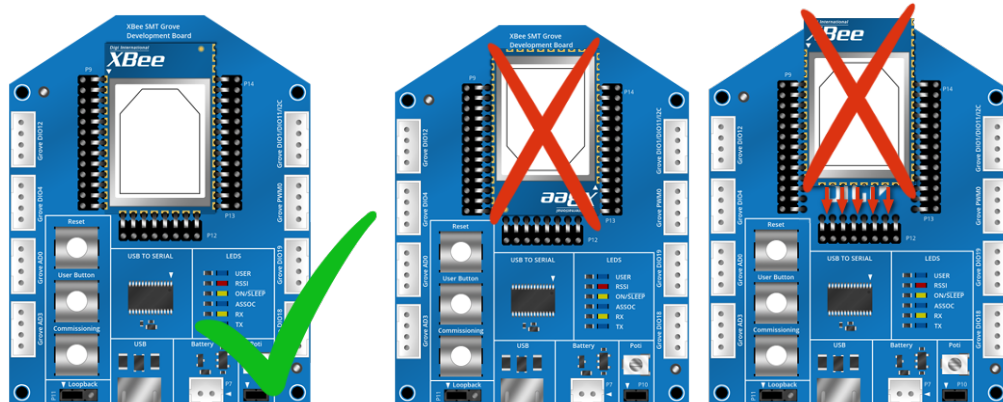


Make sure the board is NOT powered (either by the micro USB or a battery) when you plug in the XBee module.

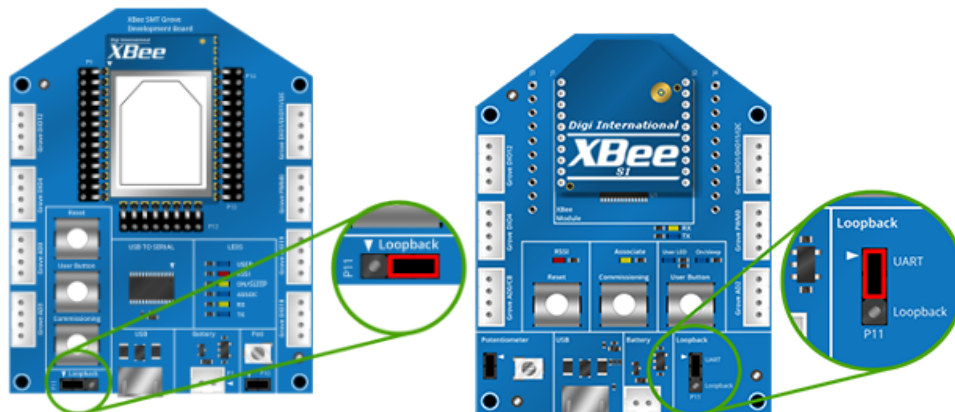
For XBee SMT modules, align all XBee pins with the spring header and carefully push the module until it is hooked to the board.



WARNING! Never insert or remove the XBee device while the power is on!



2. Once the XBee module is plugged into the board (and not before), connect the board to your computer using the micro USB cables provided.
3. Ensure the loopback jumper is in the UART position.



Unplug an XBee3 DigiMesh RF Module

To disconnect a device from the XBee Grove Development Board:

1. Disconnect the micro USB cable from the board so it is not powered.
2. Remove the device from the board socket, taking care not to bend any of the pins. The surface mount device uses spring pins rather than a socket and has a rectangular board cutout designed to help in removing the XBee3 DigiMesh RF Module.



CAUTION! Make sure the board is **not** powered when you remove the XBee3 DigiMesh RF Module.

Configure the device using XCTU

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

For instructions on downloading and using XCTU, see the [XCTU User Guide](#).

Configure remote devices

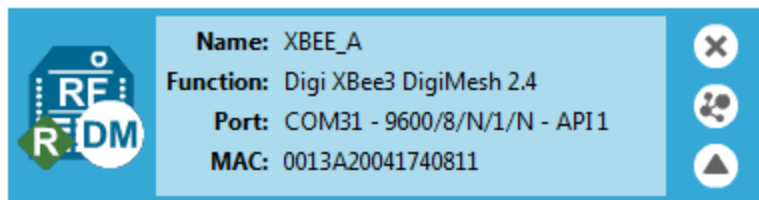
You can communicate with remote devices over the air through a corresponding local device.


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

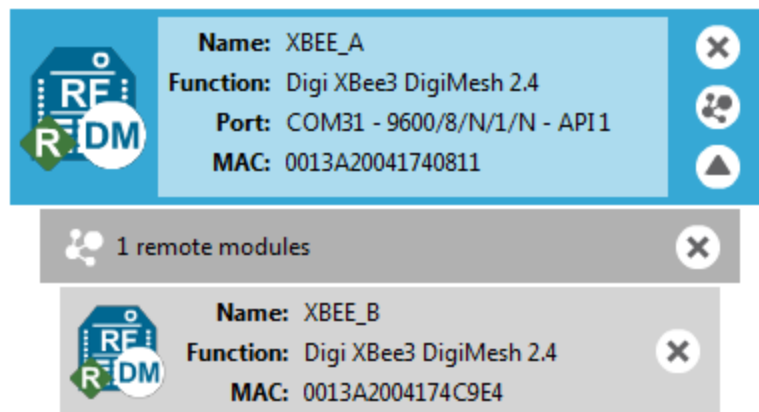
These instructions show you how to configure a remote device parameter on a remote device.

1. Add two XBee devices to XCTU.
2. Load XBee3 DigiMesh 2.4 firmware onto each device if it is not already loaded. See [How to update the firmware of your modules](#) in the *XCTU User Guide* for more information.
3. Configure the first device in API mode and name it **XBEE_A** by configuring the following parameters:
 - **ID:** 2018
 - **NI:** XBEE_A
 - **AP:** API enabled [1]
4. Configure the second device in either API or Transparent mode, and name it **XBEE_B** by configuring the following parameters:
 - **ID:** 2018
 - **NI:** XBEE_B
 - **AP:** 0 or 1

4. Disconnect XBEE_B from your computer and remove it from XCTU.
5. Connect XBEE_B to a power supply (or laptop or portable battery).
The **Radio Modules** area should look something like this.



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



9. Select the remote device **XBEE_B** to display its current configuration settings. If you want to modify a command parameter, use the radio configuration pane.
10. Click the **Write radio settings** button to apply any changes and write it to the remote device.

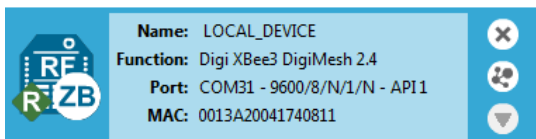
Configure the devices for a range test


1. Add two devices to XCTU.
2. Select the first module and click the **Load default firmware settings** button.
3. Configure the following parameters:
 - ID:** 2018
 - NI:** LOCAL_DEVICE
 - AP:** API Mode Enabled [1]
4. Click the **Write radio settings** button.
5. Select the other module and click the **Default firmware settings** button.
6. Configure the following parameters:
 - ID:** 2018
 - NI:** REMOTE_DEVICE

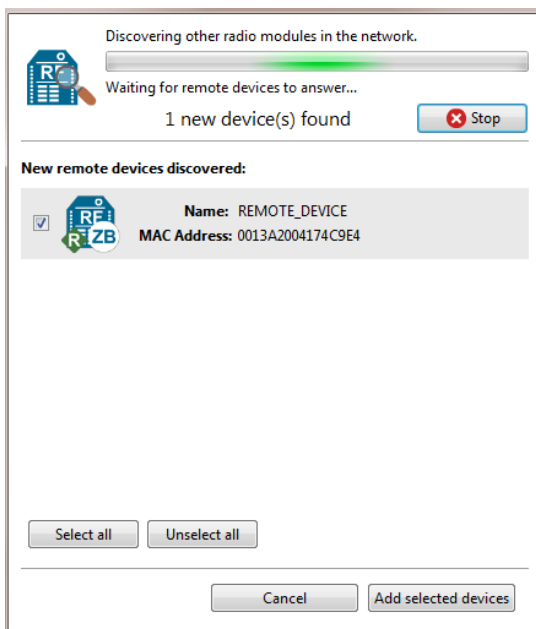
7. Click the **Write radio settings** button.
After you write the radio settings for each device, their names appear in the **Radio Modules** area. The Port indicates that the LOCAL_DEVICE is in API mode.
8. Disconnect REMOTE_DEVICE from the computer, remove it from XCTU, and connect it to a power supply, laptop, or portable battery.
9. Leave LOCAL_DEVICE connected to the computer.

Perform a range test


1. Go to the XCTU display for radio 1.

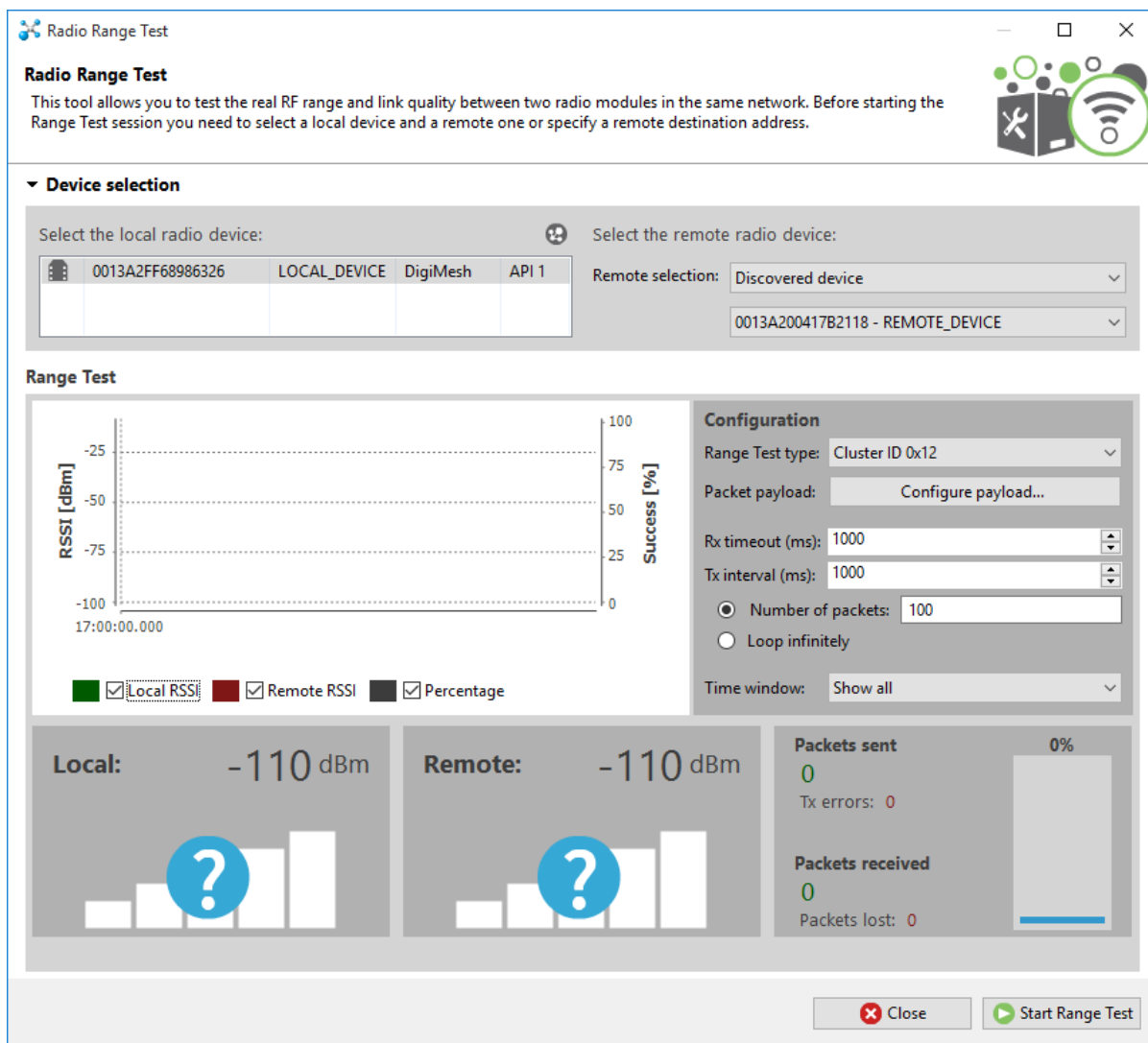


2. Click  to discover remote devices within the same network. The **Discover remote devices** dialog appears.



3. Click **Add selected devices**.

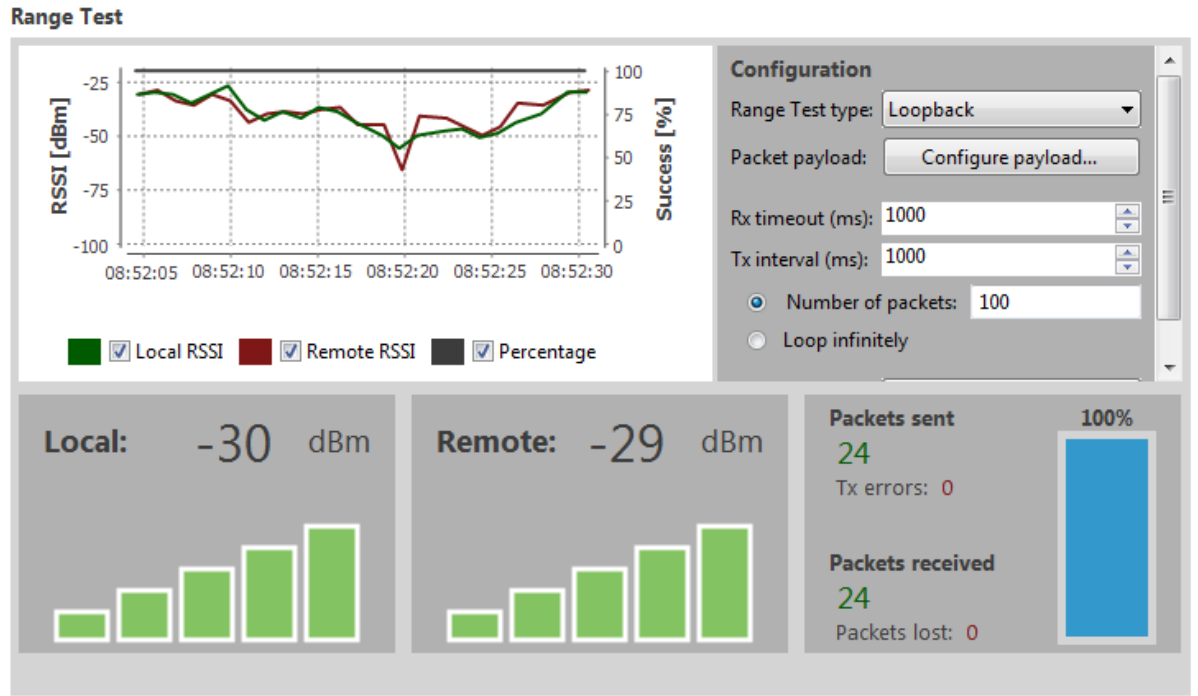
- Click  and select **Range test**. The **Radio Range Test** dialog appears.



- In the **Select the local radio device** area, select radio 1. XCTU automatically selects the **Discovered device** option, and the **Start Range Test** button is active.

6. Click **Start Range Test** to begin the range test.

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



7. Move Radio 1 around to see the resulting signal strength at different distances. You can also test different power levels by reconfiguring the [PL \(TX Power Level\)](#) parameter on both devices.

Configure the XBee3 DigiMesh RF Module

Software libraries	22
Over-the-air (OTA) firmware update	22
Custom defaults	22
XBee Network Assistant	23

Software libraries

One way to communicate with the XBee3 DigiMesh RF Module is by using a software library. The libraries available for use with the XBee3 DigiMesh 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.

Over-the-air (OTA) firmware update

The XBee3 DigiMesh RF Module supports OTA firmware updates using XCTU version 6.3.0 or higher. For instructions on performing an OTA firmware update with XCTU, see [How to update the firmware of your modules](#) in 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 \(Network Identifier\)](#)—or settings that could make remotely recovering the device difficult if they were reset—such as [ID \(Network ID\)](#).

Note You must send these commands as local AT commands, they cannot be set using [Remote AT Command Request frame - 0x17](#).

Set custom defaults

Use [%F \(Set Custom Default\)](#) to set custom defaults. When the XBee3 DigiMesh 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.

Setting a custom default that has already been set or setting a custom default to the factory default value will not reclaim the space used by the previous value. The new value takes effect but the old

value still occupies space in memory, reducing the number of custom defaults that can be set. This can be remedied by using **!C (Clear Custom Defaults)** to clear all custom defaults when changing custom default values, and by only setting custom defaults that differ from the factory defaults.

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.

Modes

Transparent operating mode	25
API operating mode	25
Command mode	25
Idle mode	27
Transmit mode	27
Receive mode	27

Transparent operating mode

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

API operating mode

API operating mode is an alternative to Transparent operating mode. API mode is a frame-based protocol that allows you to direct data on a packet basis. The device communicates UART or SPI data in packets, also known as API frames. This mode allows for structured communications with computers and microcontrollers.

The advantages of API operating mode include:

- It is easier to send information to multiple destinations
- The host receives the source address for each received data frame
- You can change parameters without entering Command mode

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 XBee3 DigiMesh 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 XBee3 DigiMesh 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.

Enter Command mode

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

If configured to operate in [Transparent operating mode](#), when entering Command mode the XBee3 DigiMesh 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 \(Guard Time\)](#).

Troubleshooting

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

There are two alternative ways to enter Command mode:

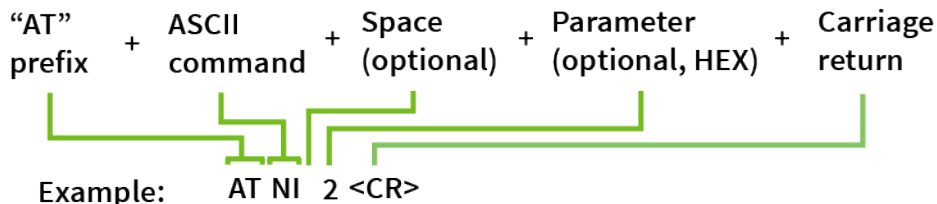
- A serial break for six seconds enters Command mode. You can issue the "break" command from a serial console, it is often a button or menu item.
- Asserting DIN (serial break) upon power up or reset enters Command mode. XCTU guides you through a reset and automatically issues the break when needed.

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 \(Network Identifier\)](#) to **My XBee**.

Multiple AT commands

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

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

Parameter format

Refer to the list of [AT commands](#) for the format of individual AT command parameters. Valid formats for hexadecimal 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 XBee3 DigiMesh RF Module responds with **OK<cr>** if successful and **ERROR<cr>** if not.

Apply command changes

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

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

Make command changes permanent

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

Send as [RE \(Restore Defaults\)](#) to wipe settings saved using **WR** back to their factory defaults, or custom defaults if you have set any.

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

Exit Command mode

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

For an example of programming the device using AT Commands and descriptions of each configurable parameter, see [AT commands](#).

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.

Transmit mode

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

Receive mode

This is the default mode for the XBee3 DigiMesh RF Module. The device is in Receive mode when it is not transmitting data. If a destination node receives a valid RF packet, the destination node transfers the data to its serial transmit buffer.

Serial communication

Serial interface	29
Serial receive buffer	29
Serial transmit buffer	29
UART data flow	29
Flow control	30

Serial interface

The XBee3 DigiMesh RF Module interfaces to a host device through a serial port. The device can communicate through its serial port:

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

Serial receive buffer

When serial data enters the XBee3 DigiMesh 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). 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.

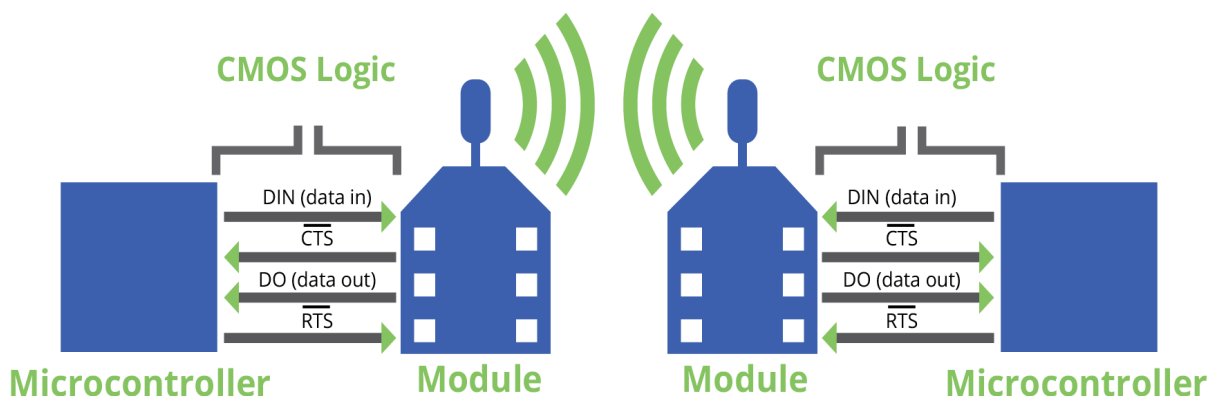
If the UART is the serial port and you enable the CTS flow control, the device alerts the external data source when the receive buffer is almost full. The host delays sending data to the device until the module asserts CTS again, allowing more data to come in.

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. Whenever the device receives data faster than it can process and transmit the data out the serial port, there is a potential of dropping data.

UART data flow

Devices that have a UART interface connect directly to the pins of the XBee3 DigiMesh 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.



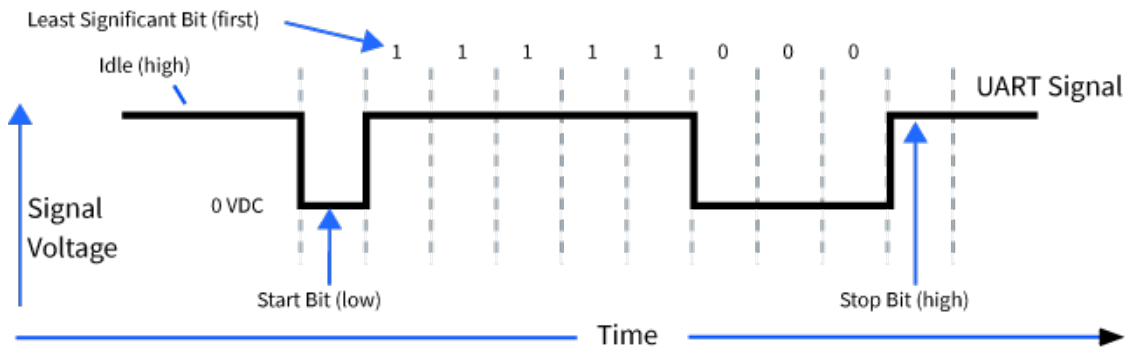
For more information about hardware specifications for the UART, see the [XBee3 Hardware Reference Manual](#).

Serial data

A device sends data to the XBee3 DigiMesh 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 XBee3 DigiMesh RF Module) with compatible settings for the baud rate, parity, start bits, stop bits, and data bits.

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

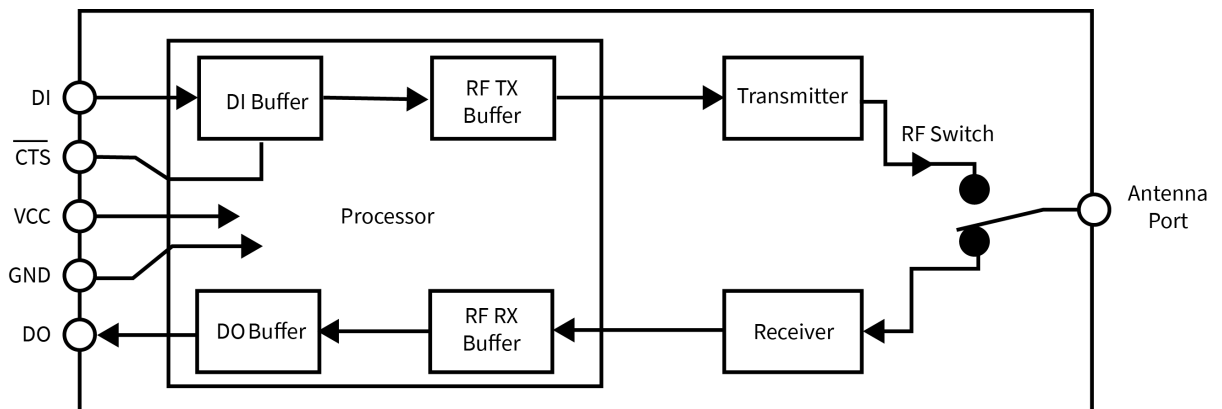


You can configure the UART baud rate, parity, and stop bits settings on the device with the **BD**, **NB**, and **SB** commands respectively. For more information, see [UART interface commands](#).

Flow control

The XBee3 DigiMesh RF Module maintains buffers to collect serial and RF data that it receives. The serial receive buffer collects incoming serial characters and holds them until the device can process them. The serial transmit buffer collects the data it receives via the RF link until it transmits that data out the serial port. The following figure shows the process of device buffers collecting received serial data.

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



Clear-to-send (CTS) flow control

If you enable $\overline{\text{CTS}}$ flow control ([D7 \(DIO7/CTS Configuration\)](#)), when the serial receive buffer is more than **FT** bytes full, the device de-asserts CTS (sets it high) to signal to the host device to stop sending serial data. The device reasserts CTS after the serial receive buffer has less than **FT** bytes in it. See [FT \(Flow Control Threshold\)](#) to configure and read this threshold.

RTS flow control

If you send [D6 \(DIO6/RTS Configuration\)](#) to enable $\overline{\text{RTS}}$ 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 $\overline{\text{RTS}}$ is de-asserted (set high) the device could send up to five characters out the UART port after RTS is de-asserted.

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

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

SPI operation

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

SPI communications	33
Full duplex operation	34
Low power operation	34
Select the SPI port	35
Force UART operation	36

SPI communications

The XBee3 DigiMesh 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.

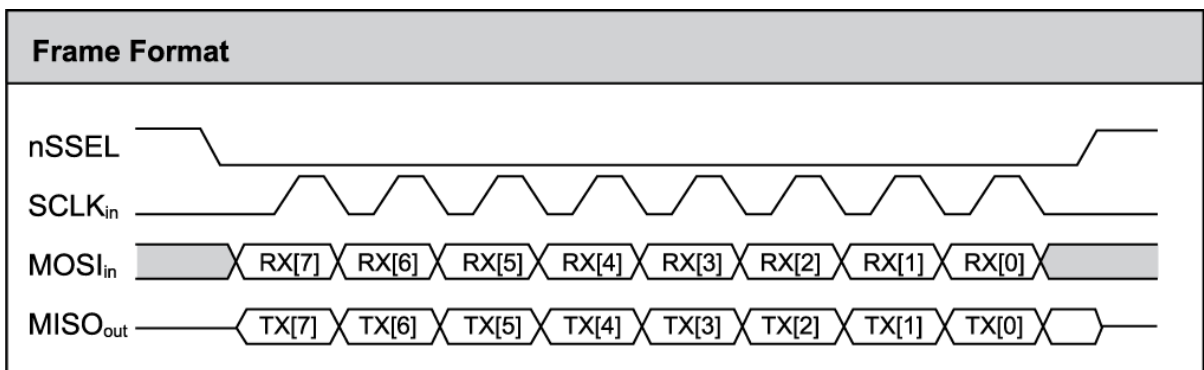
Refer to the [XBee3 Hardware Reference Guide](#) for the pinout of your device.

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_SSEL (Slave Select)	Input	Enables serial communication with the slave
SPI_ATT $\overline{\text{N}}$ (Attention)	Output	Alerts the master that slave has data queued to send. The XBee3 DigiMesh 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.



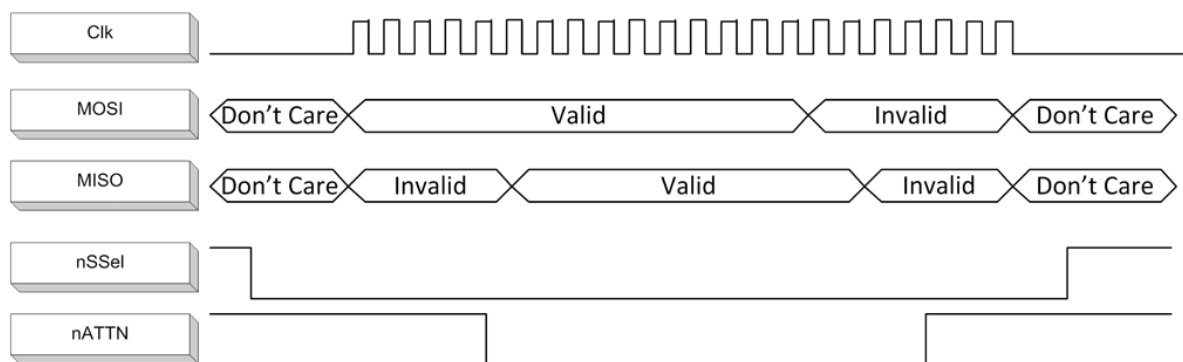
SPI mode is chip to chip communication. We do not supply a SPI communication interface on the XBee development evaluation boards included in the development kit.

Full duplex operation

When using SPI on the XBee3 DigiMesh 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.

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 **SM1** mode.

Select the SPI port

To force SPI mode on through-hole devices, hold DOUT/DIO13 low while resetting the device until SPI_ATT $\overline{\text{N}}$ 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_ATT $\overline{\text{N}}$ line to assert. The host can use this to determine that the SPI port is configured properly.

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

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

Through-hole	Micro and Surface-mount	SPI signal
D1 (DIO1/ADC1/TH_SPI_ATT $\overline{\text{N}}$ Configuration)	P9 (DIO19/SPI_ATT $\overline{\text{N}}$ Configuration)	ATT $\overline{\text{N}}$
D2 (DIO2/ADC2/TH_SPI_CLK Configuration)	P8 (DIO18/SPI_CLK Configuration)	SCLK
D3 (DIO3/ADC3/TH_SPI_SSEL Configuration)	P7 (DIO17/SPI_SSEL Configuration)	SSEL $\overline{\text{}}$
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 ATT $\overline{\text{N}}$ signal is optional—you can still use SPI mode if you disable the SPI_ATT $\overline{\text{N}}$ pin (**D1** on through-hole or **P9** on surface-mount devices).

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

If the UART is disabled and the SPI is enabled in the written configuration, then the device comes up in SPI mode without forcing it by holding DOUT low. If both the UART and the SPI are configured (**P3** (DIO13/UART_DOUT) through **P9** (DIO19/SPI_ATT $\overline{\text{N}}$ 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_SSEL $\overline{\text{}}$) 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_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_SSEL causes the SPI_MISO line to be tri-stated such that another slave device can drive it, if so desired.

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

Force UART operation

If you configure a device with only the SPI enabled and no SPI master is available to access the SPI slave port, you can recover the device to UART operation by holding DIN / CONFIG low at reset time. DIN/CONFIG forces a default configuration on the UART at 9600 baud and brings up the device in Command mode on the UART port. You can then send the appropriate commands to the device to configure it for UART operation. If you write those parameters, the device comes up with the UART enabled on the next reset.

I/O support

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

Digital I/O support	38
Analog I/O support	38
Monitor I/O lines	39
I/O sample data format	40
API frame support	41
On-demand sampling	41
Example: Command mode	41
Example: Local AT command in API mode	42
Example: Remote AT command in API mode	42
Periodic I/O sampling	43
Digital I/O change detection	44
I/O line passing	44
Digital line passing	45
Example: Digital line passing	45
Analog line passing	45
Example: Analog line passing	46
Output sample data	46
Output control	46
I/O behavior during sleep	46

Digital I/O support

Digital I/O is available on lines DIO0 through DIO12 ([D0 \(DIO0/ADC0/Commissioning Configuration\)](#) - [D9 \(DIO9/ON_SLEEP Configuration\)](#) and [P0 \(DIO10/RSSI/PWM0 Configuration\)](#) - [P4 \(DIO14/UART_DIN Configuration\)](#)). 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/ADC0/Commissioning Configuration)
DIO1	30	32	19	D1 (DIO1/ADC1/TH_SPI_ATTN Configuration)
DIO2	29	31	18	D2 (DIO2/ADC2/TH_SPI_CLK Configuration)
DIO3	28	30	17	D3 (DIO3/ADC3/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 Configuration)
DIO7	24	25	12	D7 (DIO7/CTS Configuration)
DIO8	9	10	9	D8 (DIO8/DTR/SLP_Request Configuration)
DIO9	25	26	13	D9 (DIO9/ON_SLEEP Configuration)
DIO10	7	7	6	P0 (DIO10/RSSI/PWM0 Configuration)
DIO11	8	8	7	P1 (DIO11/PWM1 Configuration)
DIO12	5	5	4	P2 (DIO12/TH_SPI_MISO Configuration)
DIO13	3	3	2	P3 (DIO13/UART_DOUT)
DIO14	4	4	3	P4 (DIO14/UART_DIN Configuration)

I\O sampling is not available for pins P5 through P9. See the [XBee3 Hardware Reference Manual](#) for full pinouts and functionality.

Analog I/O support

Analog input is available on D0 through D3. Configure these pins to **2** (ADC) to enable analog sampling. PWM output is available on P0 and P1, which can be used for [Analog line passing](#). Use [M0 \(PWM0 Duty Cycle\)](#) and [M1 \(PWM1 Duty Cycle\)](#) to set a fixed PWM level.

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

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

$$\text{ADC} / 1023 (\text{vREF}) = \text{Voltage}$$

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:

$$430 / 1023 (1.25 \text{ V}) = 525 \text{ mV}$$

Monitor I/O lines

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

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

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

The three methods to generate sample data are:

- **Periodic sample ([IR \(Sample Rate\)](#))**
 - Periodic sampling based on a timer
 - Samples are taken immediately upon wake (excluding pin sleep)
 - Sample data is sent to **DH+DL** destination address
 - Can be used with line passing
 - Requires API mode on receiver
- **Change detect ([IC \(DIO Change Detect\)](#))**
 - Samples are generated when the state of specified digital input pin(s) change
 - Sample data is sent to **DH+DL** destination address
 - Can be used with line passing
 - Requires API mode on receiver

- On-demand sample (IS (I/O Sample))
 - Immediately query the device's I/O lines
 - Can be issued locally in Command Mode
 - Can be issued locally or remotely in API mode

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

I/O sample data format

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
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.

Bytes	Name	Description
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 Data Sample Rx Indicator frame - 0x92](#) for more information on the frame's format and an example.

On-demand sampling

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

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

If you send **IS** using [Command mode](#), then the device returns a carriage return delimited list containing the I/O sample data. If **IS** is sent either locally or remotely via an API frame, the I/O sample data is presented as the parameter value in the AT command response frame ([AT Command Response frame - 0x88](#) or [Remote Command Response frame - 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 1100 0000 1100 b = DIO2, 3, 10, 11)
03	Analog channel mask, indicates which analog lines are sampled (0x03 = 0000 00 11 b = AD0, 1)
0408	Digital sample data that corresponds with the digital channel mask 0x0408 = 0000 0100 0000 1000 b = 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 [AT Command Frame - 0x08](#) or [AT Command - Queue Parameter Value frame - 0x09](#) frame:

```
7E 00 04 08 53 49 53 08
```

The device responds with a [AT Command Response frame - 0x88](#) 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 1100 0000 1100 b = 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 0100 0000 1000 b = 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 frame - 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 Command Response frame - 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

Periodic I/O sampling

Periodic sampling allows a device to take an I/O sample and transmit it to a remote device at a periodic rate.

Source

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

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

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

Destination

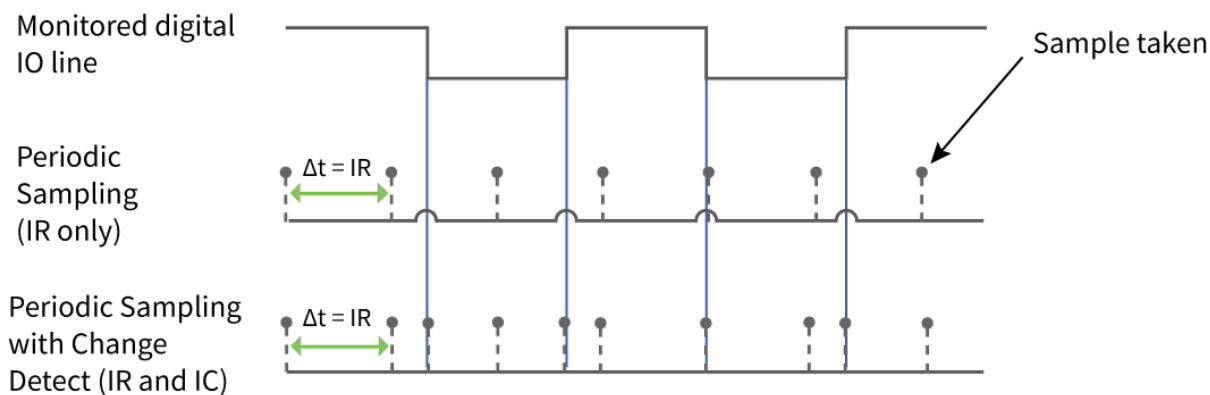
If the receiving device is operating in [API operating mode](#) the [I/O data sample](#) is emitted out of the serial port. Devices that are in [Transparent operating mode](#) discard the I/O data samples they receive unless you enable line passing.

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 \(DIO Change Detect\)](#) is a bitmask that determines which digital I/O lines to monitor for a state change. If you set one or more bits in **IC**, the device transmits an I/O sample as soon it observes a state change on the monitored digital I/O line(s) using edge detection.

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

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



Note Use caution when combining change detect sampling with [sleep modes](#). **IC** only causes a sample to be generated if a state change occurs during a wake period. If the device is sleeping when the digital transition occurs, then no change is detected and an I/O sample is not generated.

Use periodic sampling with **IR** in conjunction with **IC** in this instance, since **IR** generates an I/O sample upon wakeup and ensures that the change is properly observed.

I/O line passing

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

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

Digital line passing

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

Each digital pin has an associated timeout value. When an I/O sample is received that affects a digital output pin, the pin returns to its configured state after the timeout period expires. For pins D0 through D9, the associated timeout commands are **T0 (D0 Timeout)** through **T9 (D9 Timeout)**. For pins P0 through P4, the associated timeout commands are **Q0 (P0 Timeout)** through **Q4**.

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

Example: Digital line passing

A sampling XBee3 DigiMesh RF Module is configured with the following settings:

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

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

AT command	Parameter value
D2 (DIO2/ADC2/TH_SPI_CLK Configuration)	5 (digital output low)
T2 (D2 Output Timeout)	0x64 (10 seconds)
IA (I/O Input Address)	0013A20087654321

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

Analog line passing

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

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

Example: Analog line passing

A sampling device is configured with the following settings:

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

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

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

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

Output sample data

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

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

Output control

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

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

I/O behavior during sleep

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

Digital I/O lines

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

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

Analog and PWM I/O Lines

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

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

Networking

Network identifiers	49
Operating channels	49
Delivery methods	49
DigiMesh networking	50
Repeater/directed broadcast	52
Encryption	53
Maximum payload	53

Network identifiers

You define DigiMesh networks with a unique network identifier. Use the **ID** command to set this identifier. For devices to communicate, you must configure them with the same network identifier and the same operating channel. For devices to communicate, the **CH** and **ID** commands must be equal on all devices in the network.

The **ID** command directs the devices to talk to each other by establishing that they are all part of the same network. The **ID** parameter allows multiple DigiMesh networks to co-exist on the same physical channel.

Operating channels

The XBee3 DigiMesh RF Module operates over the 2.4 GHz band using direct sequence spread spectrum (DSSS) modulation. DSSS modulation allows the device to operate over a channel or frequency that you specify.

The 2.4 GHz frequency band defines 16 operating channels. The XBee3 DigiMesh RF Module supports all 16 channels, but output power on channel 26 on the XBee3 PRO RF Module is limited.

Use the **CH** command to select the operating channel on a device. **CH** tells the device the frequency to use to communicate.

For devices to communicate, the **CH** and **ID** commands must be equal on all devices in the network.

Note these requirements for communication:

- A device can only receive data from other devices within the same network (with the same **ID** value) and using the same channel (with the same **CH** value).
- A device can only transmit data to other devices within the same network (with the same **ID** value) and using the same channel (with the same **CH** value).

Delivery methods

The **TO** ([Transmit Options](#)) command sets the default delivery method that the device uses when in Transparent mode. In API mode, the TxOptions field of the API frame overrides the **TO** command, if non-zero.

The XBee3 DigiMesh RF Module supports three delivery methods:

- Point-to-multipoint (**TO** = 0x40).
- Repeater (directed broadcast) (**TO** = 0x80).
- DigiMesh (**TO** = 0xC0).

Point-to-multipoint

To select point-to-multipoint, set the transmit options to 0x40.

In Transparent mode, use the **TO** (Transmit Options) command to set the transmit options.

In API mode, use the Transmit Request (0x10) and Explicit Addressing Command (0x11) frames to set the transmit options. However, if the transmit options in the API frame are zero, then the transmit options in the **TO** command apply.

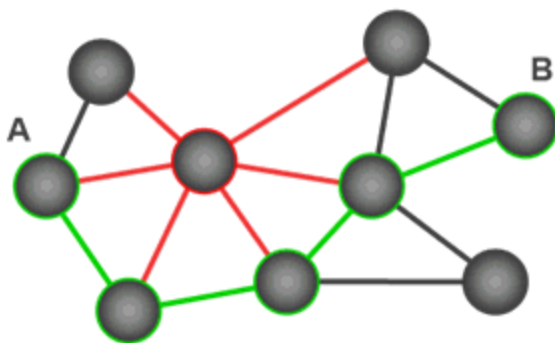
Point-to-multipoint transmissions occur between two adjacent nodes within RF range. No route discovery and no routing occur for these types of transmissions. The networking layer is entirely skipped.

Point-to-multipoint has an advantage over DigiMesh for two adjacent devices due to less overhead. However, it cannot work over multiple hops.

DigiMesh networking

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

- **Routing.** With this technique, the message is propagated along a path by hopping from node to node until it reaches its final destination.
- **Ad-hoc network creation.** This is an automated process that creates an entire network of nodes on the fly, without any human intervention.
- **Self-healing.** This process automatically figures out if one or more nodes on the network is missing and reconfigures the network to repair any broken routes.
- **Peer-to-peer architecture.** No hierarchy and no parent-child relationships are needed.
- **Quiet protocol.** Routing overhead will be reduced by using a reactive protocol similar to AODV.
- **Route discovery.** Rather than maintaining a network map, routes will be discovered and created only when needed.
- **Selective acknowledgments.** Only the destination node will reply to route requests.
- **Reliable delivery.** Reliable delivery of data is accomplished by means of acknowledgments.



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

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

Note Mesh networks use more bandwidth for routing than point-to-multipoint networks and therefore have less available for payloads.

Broadcast addressing

All of the routers in a network receive and repeat broadcast transmissions. Broadcast transmissions do not use ACKs, so the sending device sends the broadcast multiple times. By default, the sending

device sends a broadcast transmission four times. The transmissions become automatic retries without acknowledgments. This results in all nodes repeating the transmission four times as well.

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

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

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

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

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

Unicast addressing

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

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

If a device sends a unicast that uses both MAC and NWK retries and acknowledgments:

- Use MAC retries and acknowledgments for transmissions between adjacent devices in the route.
- Use NWK retries and acknowledgments across the entire route.

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

Route discovery

Route discovery is a process that occurs when:

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

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

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

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

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

Routing

A device within a mesh network determines reliable routes using a routing algorithm and table. The routing algorithm uses a reactive method derived from Ad-hoc On-demand Distance Vector (AODV). The firmware uses an associative routing table to map a destination node address with its next hop. A device sends a message to the next hop address, and the message either reaches its destination or forwards to an intermediate router that routes the message on to its destination.

If a message has a broadcast address, it is broadcast to all neighbors, then all routers that receive the message rebroadcast the message **MT**+1 times. Eventually, the message reaches the entire network.

Packet tracking prevents a node from resending a broadcast message more than **MT**+1 times. This means that a node that relays a broadcast will only relay it after it receives it the first time and it will discard repeated instances of the same packet.

Routers

You can use the **CE** command to configure devices in a DigiMesh network to act as routers or end devices. All devices in a DigiMesh network act as routers by default. Any devices that you configure as routers actively relay network unicast and broadcast traffic.

Repeater/directed broadcast

All of the routers in a network receive and repeat directed broadcast transmissions. Because it does not use ACKs, the originating node sends the broadcast multiple times. By default a broadcast transmission is sent four times—the extra transmissions become automatic retries without acknowledgments. This results in all nodes repeating the transmission four times. Sending frequent broadcast transmissions can quickly reduce the available network bandwidth, so use broadcast transmissions sparingly.

MAC layer

The MAC layer is the building block that is used to build repeater capability. To implement Repeater mode, we use a network layer header that comes after the MAC layer header in each packet. In this network layer there is additional packet tracking to eliminate duplicate broadcasts.

In this delivery method, the device sends both unicast and broadcast packets out as broadcasts that are always repeated. All repeated packets are sent to every device. The devices that receive the broadcast send broadcast data out their serial port.

When a device sends a unicast, it specifies a destination address in the network header. Then, only the device that has the matching destination address sends the unicast out its serial port. This is called a directed broadcast.

Any node that has a **CE** parameter set to router rebroadcasts the packet if its **BH** (broadcast hops) or broadcast radius values are not depleted. If a node has already seen a repeated broadcast, it ignores the broadcast.

The **BH** parameter sets the maximum number of hops that a broadcast is repeated, but there are two special cases. If **BH** is **0** or if **BH** is **> NH**, then **NH** specifies the maximum hops for broadcasts instead.

By default the **CE** parameter is set to route all broadcasts. As such, all nodes that receive a repeated packet will repeat it. If you change the **CE** parameter, you can limit which nodes repeat packets, which helps dense networks from becoming overly congested while packets are being repeated.

Transmission timeout calculations for Repeater/directed broadcast mode are the same as for DigiMesh broadcast transmissions.

The MAC layer is the building block that is used to build repeater capability. To implement Repeater mode, we use a network layer header that comes after the MAC layer header in each packet. In this network layer there is additional packet tracking to eliminate duplicate broadcasts.

Encryption

XBee3 DigiMesh provides greater security against replay attacks and attempts to determine the plaintext. The XBee3 DigiMesh RF Module performs Counter (CTR) mode encryption instead of Electronic Codebook (ECB) mode encryption. Since the counter is passed over-the-air (OTA) and changes with each frame, the same text is always encrypted differently and there are no known attacks to determine the plaintext from the ciphertext.

A side effect of this implementation is that the maximum payload is reduced by the size of the counter (8 bytes). Therefore, no frames can exceed 65 bytes with encryption enabled. The maximum payload is still 73 bytes with encryption disabled.

Also effective with XBee3 DigiMesh, the key is 256 bits rather than 128 bits. 256 bits is 32 bytes. Since the key is entered with ASCII HEX characters in Command mode, up to 64 ASCII HEX characters may be entered for the **KY** command.

For compatibility with nodes in the same network that do not support CTR mode encryption, [C8 \(Compatibility Options\)](#) bit 2 was introduced to enable the 128-bit key with ECB mode encryption as supported previously. In this case, only the last 32 ASCII HEX characters of the key are used, even if more characters were previously entered for the key.

Maximum payload

DigiMesh uses the 802.15.4 PHY layer including a 2-byte CRC at the end of the frame. This reduces the size of each frame to 125 bytes. After the MAC header, the NWK header, and the APP header are included at the beginning of the packet, the remaining space is 73 bytes for payload. If CTR mode encryption is enabled, this number is further reduced to 65 bytes. The best way to determine the maximum payload is to read [NP \(Maximum Packet Payload Bytes\)](#).

These maximums only apply in API mode. If you attempt to send an API packet with a larger payload than specified, the device responds with a Transmit Status frame (0x89) with the Status field set to **74** (Data payload too large).

In Transparent mode, the firmware splits the data as necessary to cope with maximum payloads.

Network commissioning and diagnostics

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

Local configuration	55
Remote configuration	55
Build aggregate routes	56
RSSI indicators	60
Associate LED	60
The Commissioning Pushbutton	60
Node discovery	62

Local configuration

You can configure devices locally using serial commands in Command mode or API mode, or remotely using remote AT commands. Devices that are in API mode can send configuration commands to set or read the configuration settings of any device in the network.

Remote configuration

When you do not have access to the device's serial port, you can use a separate device in API mode to remotely configure it. To remotely configure devices, use the following steps.

Send a remote command

To send a remote command, populate the [Remote AT Command Request frame - 0x17](#) with:

1. The 64-bit address of the remote device.
2. The correct command options value.
3. Optionally, the command and parameter data.
4. If you want a command response, set the Frame ID field to a non-zero value.

XCTU has a Frames Generator tool that can assist you with building and sending a remote AT frame; see [Frames generator tool](#) in the *XCTU User Guide*.

Apply changes on remote devices

When you use remote commands to change the command parameter settings on a remote device, you must apply the parameter changes or they do not take effect. For example, if you change the **BD** parameter, the actual serial interface rate does not change on the remote device until you apply the changes. You can apply the changes using remote commands in one of three ways:

1. Set the apply changes option bit in the API frame.
2. Send an **AC** command to the remote device.
3. Send the **WR** command followed by the **FR** command to the remote device to save the changes and reset the device.

Remote command response

If a local device sends a command request to a remote device, and the API frame ID is non-zero, the remote device sends a remote command response transmission back to the local device.

When the local device receives a remote command response transmission, it sends a remote command response API frame out its UART. The remote command response indicates:

1. The status of the command, which is either success or the reason for failure.
2. In the case of a command query, it includes the register value.

The device that sends a remote command does not receive a remote command response frame if:

1. It could not reach the destination device.
2. You set the frame ID to 0 in the remote command request.

Build aggregate routes

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

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

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

The **AG** command requires a 64-bit parameter. The parameter indicates the current value of the **DH/DL** registers on a device; typically you should replace this value with the 64-bit address of the node sending the **AG** broadcast. However, if you do not want to update the **DH/DL** of the device receiving the **AG** broadcast you can use the invalid address of 0xFFFFE. The receiving nodes that are configured in API mode output an Aggregator Update API frame (0x8E) if they update their **DH/DL** address; for a description of the frame, see [Aggregate Addressing Update frame - 0x8E](#).

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

DigiMesh routing examples

Example one

In a scenario where you deploy a network, and then you want to update the **DH** and **DL** registers of all the devices in the network so that they use the MAC address of the aggregator node, which has the MAC address 0x0013A200 4052C507, you could use the following technique.

1. Deploy all devices in the network with the default **DH/DL** of 0xFFFF.
2. Serially, send an ATAGFFFF command to the aggregator node so it sends the broadcast transmission to the rest of the nodes.

All the nodes in the network that receive the **AG** broadcast set their **DH** to 0x0013A200 and their **DL** to 0x4052C507. These nodes automatically build a route to the aggregator node.

Example two

If you want all of the nodes in the network to build routes to an aggregator node with a MAC address of 0x0013A200 4052C507 without affecting the **DH** and **DL** registers of any nodes in the network:

1. Send the ATAGFFFE command to the aggregator node. This sends an **AG** broadcast to all of the nodes in the network.
2. All of the nodes internally update only their routing table information to contain a route to the aggregator node.

3. None of the nodes update their **DH** and **DL** registers because none of the registers are set to the 0xFFFE address.

Replace nodes

You can use the **AG** command to update the routing table and **DH/DL** registers in the network after you replace a device. To update only the routing table information without affecting the **DH** and **DL** registers, use the process in example two, above.

To update the **DH** and **DL** registers of the network, use the following example.

Example

This example shows how to cause all devices to update their **DH** and **DL** registers to the MAC address of the sending device. In this case, assume you are using a device with a serial number of 0x0013A200 4052C507 as a network aggregator, and the sending device has a MAC address of 0x0013A200 F5E4D3B2. To update the **DH** and **DL** registers to the sending device's MAC address:

1. Replace the aggregator with 0x0013A200 F5E4D3B2.
2. Send the ATAG0013A200 4052C507 command to the new device.

Test links between adjacent devices

It often helps to test the quality of a link between two adjacent modules in a network. You can use the Test Link Request Cluster ID to send a number of test packets between any two devices in a network. To clarify the example, we refer to "device A" and "device B" in this section.

To request that device B perform a link test against device A:

1. Use device A in API mode (**AP** = 1) to send an Explicit Addressing Command (0x11) frame to device B.
2. Address the frame to the Test Link Request Cluster ID (0x0014) and destination endpoint: 0xE6.
3. Include a 12-byte payload in the Explicit Addressing Command frame with the following format:

Number of bytes	Field name	Description
8	Destination address	The address the device uses to test its link. For this example, use the device A address.
2	Payload size	The size of the test packet. Use the NP command to query the maximum payload size for the device.
2	Iterations	The number of packets to send. This must be a number between 1 and 4000.

4. Device B should transmit test link packets.
5. When device B completes transmitting the test link packets, it sends the following data packet to device A's Test Link Result Cluster (0x0094) on endpoint (0xE6).
6. Device A outputs the following information as an API Explicit RX Indicator (0x91) frame:

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

Example

Suppose that you want to test the link between device A (**SH/SL** = 0x0013A200 40521234) and device B (**SH/SL**=0x0013A 200 4052ABCD) by transmitting 1000 40-byte packets:

Send the following API packet to the serial interface of device A.

In the following example packet, whitespace marks fields, bold text is the payload portion of the packet:

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

When the test is finished, the following API frame may be received:

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

This means:

- 999 out of 1000 packets were successful.
- The device made 100 retries.
- **RR** = 10.
- maxRSSI = -80 dBm.
- minRSSI = -83 dBm.
- avgRSSI = -82 dBm.

If the Result field does not equal zero, an error has occurred. Ignore the other fields in the packet.

If the Success field equals zero, ignore the RSSI fields.

The device that sends the request for initiating the Test link and outputs the result does not need to be the sender or receiver of the test. It is possible for a third node, "device C", to request device A to perform a test link against device B and send the results back to device C to be output. It is also possible for device B to request device A to perform the previously mentioned test. In other words, the frames can be sent by either device A, device B or device C and in all cases the test is the same: device A sends data to device B and reports the results.

Trace route option

In many networks, it is useful to determine the route that a DigiMesh unicast takes to its destination, particularly when you set up a network or want to diagnose problems within a network.

Note Because of the large number of Route Information Packet frames that a unicast with trace route enabled can generate, we suggest you only use the trace route option for occasional diagnostic purposes and not for normal operations.

The Transmit Request (0x10 and 0x11) frames contain a trace route option, which transmits routing information packets to the originator of the unicast using the intermediate nodes.

When a device sends a unicast with the trace route option enabled, the unicast transmits to its destination devices, which forward the unicast to its eventual destination. The destination device transmits a Route Information Packet (0x8D) frame back along the route to the unicast originator.

The Route Information Packet frame contains:

- Addressing information for the unicast
- Addressing information for the intermediate hop
- Timestamp
- Other link quality information

For a full description of the Route Information Packet frame, see [Route Information Packet frame - 0x8D](#).

Trace route example

Suppose that you successfully unicast a data packet with trace route enabled from device A to device E, through devices B, C, and D. The following sequence would occur:

- After the data packet makes a successful MAC transmission from device A to device B, device A outputs a Route Information Packet frame indicating that the transmission of the data packet from device A to device E was successful in forwarding one hop from device A to device B.
- After the data packet makes a successful MAC transmission from device B to device C, device B transmits a Route Information Packet frame to device A. When device A receives the Route Information packet, it outputs it over its serial interface.
- After the data packet makes a successful MAC transmission from device C to device D, device C transmits a Route Information Packet frame to device A (through device B). When device A receives the Route Information packet, it outputs it over its serial interface.
- After the data packet makes a successful MAC transmission from device D to device E, device D transmits a Route Information Packet frame to device A (through device C and device B). When device A receives the Route Information packet, it outputs it over its serial interface.

There is no guarantee that Route Information Packet frames will arrive in the same order as the route taken by the unicast packet. On a weak route, it is also possible for the transmission of Route Information Packet frames to fail before arriving at the unicast originator.

NACK messages

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

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

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

RSSI indicators

The received signal strength indicator (RSSI) measures the amount of power present in a radio signal. It is an approximate value for signal strength received on an antenna.

You can use the **DB** command to measure the RSSI on a device. **DB** returns the RSSI value measured in -dBm of the last packet the device received. This number can be misleading in multi-hop DigiMesh networks. The **DB** value only indicates the received signal strength of the last hop. If a transmission spans multiple hops, the **DB** value provides no indication of the overall transmission path, or the quality of the worst link, it only indicates the quality of the last link.

To determine the **DB** value in hardware:

1. Set **PO** to 1 to enable the RSSI pulse-width modulation (PWM) functionality.
2. Use the DIO10/RSSI/PWM0 module pin (Micro pin 7/SMT pin 7/TH pin 6). When the device receives data, it sets the RSSI PWM duty cycle to a value based on the RSSI of the packet it receives.

This value only indicates the quality of the last hop of a multi-hop transmission. You could connect this pin to an LED to indicate if the link is stable or not.

Associate LED

The Associate pin (Micro pin 26/SMT pin 28) provides an indication of the device's status. To take advantage of these indications, connect an LED to the Associate pin.

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

The pin functions as a power indicator.

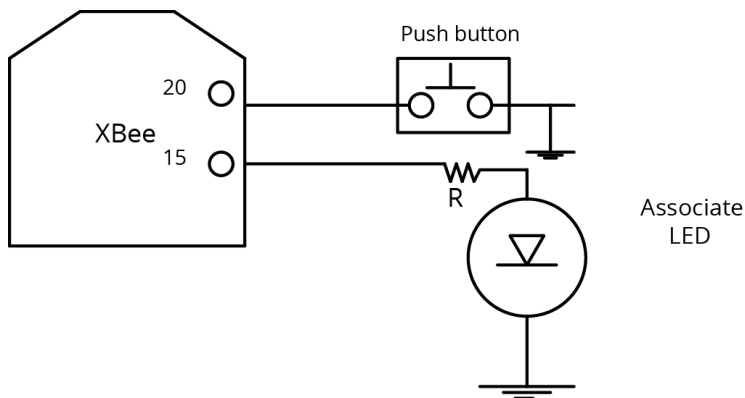
Use the **LT** command to override the blink rate of the Associate pin. If you set **LT** to 0, the device uses the default blink time of 250 ms.

The following table describes the Associate LED functionality.

LED Status	Meaning
On, blinking	The device has power and is operating properly

The Commissioning Pushbutton

The XBee3 DigiMesh RF Module supports a set of commissioning and LED functions to help you deploy and commission devices. These functions include the Commissioning Pushbutton definitions and the associated LED functions. The following diagram shows how the hardware can support these features.



To support the Commissioning Pushbutton and its associated LED functions, connect a pushbutton and an LED to device pins 20 and 15 respectively.

Definitions

To enable the Commissioning Pushbutton functionality on pin 20, set the **DO** command to 1. The functionality is enabled by default.

You must perform multiple button presses within two seconds.

The following table provides the pushbutton definitions.

Button presses	Action
1	Immediately sends a Node Identification broadcast transmission. All devices that receive this transmission blink their Associate LED rapidly for one second. All devices in API operating mode that receive this transmission send a Node Identification Indicator frame (0x95) out their UART.
1	If the device is configured for asynchronous sleep, this wakes it for 30 seconds. Immediately sends a Node Identification broadcast transmission. All devices that receive this transmission blink their Associate LED rapidly for one second. All devices in API operating mode that receive this transmission send a Node Identification Indicator frame (0x95) out their UART.
4	Sends an RE command to restore device parameters to default values.

Use the Commissioning Pushbutton

Use the **CB** command to simulate button presses in software. Send **CB** with a parameter set to the number of button presses to perform. For example, if you send **ATCB1**, the device performs the action (s) associated with a single button press.

[Node Identification Indicator frame - 0x95](#) is similar to [Remote Command Response frame - 0x97](#) – it contains the device’s address, node identifier string (**NI** command), and other relevant data. All devices in API operating mode that receive the Node Identification Indicator frame send it out their UART as a Node Identification Indicator frame.

Node discovery

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

Commands	Syntax	Description
Node Discovery	ND	Seeks to discover all nodes in the network (on the current Network ID).
Directed Node Discovery	ND <NI String>	Seeks to discover if a particular node named <NI String> is found in the network.
Destination Node	DN <NI String>	Sets DH/DL to point to the MAC address of the node whose <NI String> matches.

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

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

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

Discover all the devices on a network

You can use the **ND** (Network Discovery) command to discover all devices on a network. When you send the **ND** command:

1. The device sends a broadcast **ND** command through the network.
2. All devices that receive the command send a response that includes their addressing information, node identifier string and other relevant information. For more information on the node identifier string, see [NI \(Network Identifier\)](#).

ND is useful for generating a list of all device addresses in a network.

When a device receives the network discovery command, it waits a random time before sending its own response. You can use the **NT** command to set the maximum time delay on the device that you use to send the **ND** command.

- The device that sends the **ND** includes its **NT** setting in the transmission to provide a random delay window for all devices in the network. When devices respond at random intervals during the **NT** window, fewer collisions occur and more responses can be obtained.
- The default **NT** value is 0x82 (13 seconds).

Directed node discovery

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

In Transparent mode, the requesting node outputs an extra carriage return following the response from the designated node and the command terminates; it is then ready to accept a new AT

command. In the event that the requested node does not exist or is too slow to respond, the requesting node outputs an ERROR response after **N?** expires.

In API mode, the response from the requesting node will be output in API mode and the command will terminate immediately. If no response comes from the requested node, the requesting node outputs an error response in API mode after **N?** expires. The device's software assumes that each node has a unique **NI** string.

The directed node discovery command terminates after the first node with a matching **NI** string responds. If that **NI** string is duplicated in multiple nodes, the first responding node may not always be the same node or the desired node.

Destination Node

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

If an **NI** string parameter is not provided, the **DN** command terminates immediately with an error. If a node with the given **NI** string does not respond, the **DN** command terminates with an error after **N?** times out.

In Transparent mode, unlike **ND** (with or without an **NI** string), **DN** does not cause the information from the responding node to be output; rather it simply sets **DH/DL** to the address of the responding node.

In API mode, the response from the requesting node outputs in API mode and the command terminates immediately. If no response comes from the requested node, the requesting node outputs an error response in API mode after **N?** expires.

The device's software assumes that each node has a unique **NI** string. The directed destination node command terminates after the first node with a matching **NI** string responds. If that **NI** string is duplicated in multiple nodes, **DH/DL** may not be set to the desired value.

Discover devices within RF range

The **FN** (Find Neighbor) command works the same as the **ND** (Node Discovery) except that it is limited to neighboring devices (devices that are only one hop away). See [FN \(Find Neighbors\)](#) for details.

- You can use the **FN** (Find Neighbors) command to discover the devices that are immediate neighbors (within RF range) of a particular device.
- **FN** is useful in determining network topology and determining possible routes.

You can send **FN** locally on a device in Command mode or you can use a local [AT Command Frame - 0x08](#).

To use **FN** remotely, send the target node a [Remote AT Command Request frame - 0x17](#) using **FN** as the name of the AT command.

The device you use to send **FN** transmits a zero-hop broadcast to all of its immediate neighbors. All of the devices that receive this broadcast send an RF packet to the device that transmitted the **FN** command. If you sent **FN** remotely, the target devices respond directly to the device that sent the **FN** command. The device that sends **FN** outputs a response packet in the same format as an [AT Command Response frame - 0x88](#).

Sleep support

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

Sleep modes	65
Sleep parameters	66
Sleep current	66
Sleep pins	66
Sleep conditions	67

Sleep modes

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

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

The following table shows the sleep mode configurations.

Sleep mode	Description
SM 0	No sleep
SM 1	Pin sleep
SM 4	Cyclic sleep
SM 5	Cyclic sleep with pin wake-up

Pin Sleep mode (SM = 1)

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

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

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

Cyclic Sleep mode (SM = 4)

The Cyclic Sleep modes allow devices to periodically check for RF data. When the **SM** parameter is set to **4**, the XBee3 DigiMesh RF Module is configured to sleep, then wakes once per cycle to check for data from a coordinator. The Cyclic Sleep Remote sends a poll request to the coordinator at a specific interval set by **SP (Sleep Time)**. The coordinator transmits any queued data addressed to that specific remote upon receiving the poll request.

If no data is queued for the remote, the messaging coordinator does not transmit and the remote returns to sleep for another cycle. If queued data is transmitted back to the remote, it stays awake to allow for back and forth communication until the **ST (Wake Time)** timer expires. You can also set **SO (Sleep Options)** bit 8 to force the device to always wake for the full **ST** time.

If configured, **CTS** goes low each time the remote wakes, allowing for communication initiated by the remote host if desired. If **ON_SLEEP** is configured it goes high (ON) after **SN (Number of Sleep Periods)** sleep periods. Change **SN** to allow external circuitry to sleep for longer periods if no data is received.

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

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

The SLEEP_RQ pin is level-triggered. The device wakes when a low is detected then sets $\overline{\text{CTS}}$ low as soon as it is ready to transmit or receive. The device stays awake as long as SLEEP_RQ is low; once SLEEP_RQ goes high the device returns to cyclic sleep operation.

Any activity resets the [ST \(Wake Time\)](#) timer, so the device goes back to sleep only after there is no activity for the duration of the timer.

Sleep parameters

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

- [SM command](#)
- [SN \(Number of Sleep Periods\)](#)
- [SO \(Sleep Options\)](#)
- [ST \(Wake Time\)](#)
- [SP \(Sleep Time\)](#)
- [WH \(Wake Host Delay\)](#)

Sleep current

The following table shows the sleep current during the XBee3 DigiMesh RF Module sleep modes.

Sleep mode	SM command setting	Sleep current
Pin sleep	1	<1 μA @ 25 °C
Cyclic sleep	4	<1 μA @ 25 °C
Cyclic sleep with pin wake-up	5	<1 μA @ 25 °C

You can make devices use low sleep current by driving PWM outputs high during sleep and by using internal pull-ups/pull-downs on disabled/unused pins.

The sleep pins are set up for sleeping as specified in the following section, [Sleep pins](#). Additionally, pins that are outputs (other than PWM outputs) continue to output the same levels during sleep. Normally, this means that pins configured for output high or low will output high or low accordingly. However, if I/O line passing overrides the output, the output level is maintained during the sleep time.

Sleep pins

The following table describes the three external device pins associated with sleep. Refer to the XBee3 Hardware Reference Guide. See the [XBee3 RF Module Hardware Reference Manual](#) for the pinout of your device.

Pin name	Pin number	Description
SLEEP_ RQ	Micro pin 9/SMT pin 10	For SM = 1 , high puts the device to sleep and low wakes it up. For SM = 5 , a high to low transition wakes the device up for ST time. The device ignores a low to high transition in SM = 5 .
$\overline{\text{CTS}}$	Micro pin 24/SMT pin 25	If D7 = 1 , high indicates that the device is asleep and low indicates that it is awake and ready to receive serial data.
$\overline{\text{ON_SLEEP}}$	Micro pin 25/SMT pin 26	Low indicates that the device is asleep and high indicates that it is awake and ready to receive serial data.

Sleep conditions

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

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

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

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

AT commands

Networking commands	69
DigiMesh Addressing commands	75
DigiMesh configuration commands	77
Diagnostic commands - addressing timeouts	79
Security commands	80
RF interfacing commands	81
MAC diagnostics commands	82
Sleep settings commands	84
UART interface commands	86
Command mode options	89
UART pin configuration commands	90
SPI interface commands	92
I/O settings commands	95
I/O sampling commands	104
I/O line passing commands	106
Diagnostics – Firmware/Hardware Information	110
Memory access commands	112
Custom default commands	113

Networking commands

The following commands affect the DigiMesh network.

CH (Operating Channel)

Set or read the operating channel devices used to transmit and receive data.

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

The command uses IEEE 802.15.4 channel numbers.

Parameter range

0xB - 0x1A

Default

0xC (channel 12)

ID (Network ID)

Set or read the user network identifier.

Devices must have the same network identifier to communicate with each other.

Parameter range

0 - 0xFFFF

Default

0x7FFF

CE (Routing / Messaging Mode)

The routing mode of the XBee3 DigiMesh RF Module.

A routing device forwards broadcasts and route discoveries for unicasts. A non-routing device does neither.

Indirect Messaging Coordinator

Device will not transmit point to multi-point unicasts until an end device requests them.

Indirect Messaging Poller

Device will periodically poll a coordinator for messages.

Parameter range

0 - 6

Parameter	Description	Routes packets
0	Standard router	Yes
1	Indirect message coordinator	Yes

Parameter	Description	Routes packets
2	Non-routing device	No
3	Non-routing coordinator	No
4	Indirect message poller	Yes
5	N/A	N/A
6	Non-routing poller	No

Default

0

C8 (Compatibility Options)

Sets or displays the operational compatibility with a legacy DigiMesh 2.4 device (S1 or S2C hardware). This parameter should only be set when operating in a mixed network that contains XBee Series 1 or XBee S2C devices.

Parameter range

0, 4

Bit field:

Bit	Meaning	Setting	Description
2	TX compatibility	0	When encryption is enabled, AES Counter mode is used with a 256-bit key.
		1	When encryption is enabled AES ECB mode is used with a 128-bit key. This is compatible with legacy versions of DigiMesh 2.4.

Default

0

NI (Network Identifier)

Stores the node identifier string for a device, which is a user-defined name or description of the device. This can be up to 20 ASCII characters.

XCTU prevents you from exceeding the string limit of 20 characters for this command. If you are using another software application to send the string, you can enter longer strings, but the software on the device returns an error.

Use the **ND** (Network Discovery) command with this string as an argument to easily identify devices on the network.

The **DN** command also uses this identifier.

Parameter range

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

Default

0x20 (an ASCII space character)

ND (Network Discover)

Discovers and reports all of the devices it finds on a network. If you send **ND** through a local or remote API frame, each network node returns a separate AT Command Response (0x88) or Remote Command Response (0x97) frame, respectively.

The command reports the following information after a jittered time delay.

```
PARENT_NETWORK ADDRESS<CR> (2 Bytes) (always 0xFFFE)
SH<CR>
SL<CR>
NI<CR> (Variable length)
PARENT_NETWORK ADDRESS (2 Bytes) <CR>
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.)
<CR>
```

If you send the **FN** command in Command mode, after (**NT***100) ms + overhead time, the command ends by returning a carriage return, represented by <CR>.

The **ND** command accepts an **NI** (Node Identifier) as an argument. For more details, see [Directed node discovery](#).

Broadcast an **ND** command to the network. If the command includes an optional node identifier string parameter, only those devices with a matching **NI** string respond without a random offset delay. If the command does not include a node identifier string parameter, all devices respond with a random offset delay.

The **NT** setting determines the range of the random offset delay. The **NO** setting sets options for the Node Discovery.

For more information about options that affect the behavior of the **ND** command Refer to [NO \(Network Discovery Options\)](#) for options which affect the behavior of the **ND** command.



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

Parameter range

20-byte printable ASCII string

Default

N/A

DN (Discover Node)

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

The following events occur after **DN** discovers the destination node:

When **DN** is sent in Command mode:

1. The requesting node sets **DL** and **DH** to the address of the device with the matching **NI** string.
2. The requesting node returns **OK** (or ERROR).
3. If the requesting node returns **OK** (node found), it exits Command mode immediately with **DH/DL** set to the node that is found so that the next serial input is sent to the node designated by the **DN** parameter.
4. If the requesting node returns **ERROR**, (node not found), it remains in Command mode, allowing you to enter further commands.

When **DN** is sent as a local [AT Command Frame - 0x08](#):

1. The requesting node returns 0xFFFE followed by its 64-bit extended addresses in an [AT Command Response frame - 0x88](#).
2. If there is no response from a module within (**N?*** 100) milliseconds or you do not specify a parameter (by leaving it blank), the requesting node returns an ERROR message.

Parameter range

20-byte ASCII string

Default

N/A

FN (Find Neighbors)

Discovers and reports all devices found within immediate (1 hop) RF range. **FN** reports the following information for each device it discovers:

```

MY<CR> (always 0xFFFE)
SH<CR>
SL<CR>
NI<CR> (Variable length)
PARENT_NETWORK_ADDRESS<CR> (2 Bytes) (always 0xFFFE)
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.)
<CR>

```

If you send the **FN** command in Command mode, after (**NT***100) ms + overhead time, the command ends by returning a carriage return, represented by <CR>.

If you send the **FN** command through a local AT Command (0x08) or remote AT command (0x17) API frame, each response returns as a separate AT Command Response (0x88) or Remote Command Response (0x97) frame, respectively. The data consists of the bytes in the previous list without the carriage return delimiters. The **NI** string ends in a 0x00 null character.

FN accepts a **NI** (Node Identifier) as an argument.

See [Find specific neighbor](#) for more details.

Parameter range

0 to 20 ASCII characters

Default

N/A

NT (Network Discovery Back-off)

The **ND** and **FN** commands use **NT**. The read-only **N?** command increases and decreases with **NT**.

Parameter range

0x20 - 0x2EE0 (x 100 ms)

Default

0x82 (13 seconds)

NO (Network Discovery Options)

Set or read the network discovery options value for [ND \(Network Discover\)](#) on a particular device. The options bit field value changes the behavior of the **ND** command and what optional values the local device returns when it receives an **ND** command or API Node Identification Indicator (0x95) frame.

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

Parameter range

0x0 - 0x7 (bit field)

Option	Description
0x01	Append the DD (Digi Device Identifier) value to ND responses or API node identification frames.
0x02	Local device sends ND response frame out the serial interface when ND is issued.
0x04	Append the RSSI of the last hop to ND , FN , and responses or API node identification frames.

Default

0x0

NP (Maximum Packet Payload Bytes)

Reads the maximum number of RF payload bytes that you can send in a transmission.

The XBee3 DigiMesh RF Module firmware returns a fixed number of bytes: 0x49 = 73 bytes without encryption, 65 bytes with encryption.

Note **NP** returns a hexadecimal value. For example, if **NP** returns 0x41, this is equivalent to 65 bytes.

Parameter range

[read-only]

Default

N/A

DigiMesh Addressing commands

The following commands affect the source and destination addressing for the device.

SH (Serial Number High)

Displays the upper 32 bits of the unique IEEE 64-bit address assigned to the XBee in the factory. The 64-bit source address is always enabled. This value is read-only and it never changes.

Parameter range

0 - 0xFFFFFFFF [read-only]

Default

Set in the factory

SL (Serial Number Low)

Displays the lower 32 bits of the unique IEEE 64-bit RF address assigned to the XBee in the factory. The 64-bit source address is always enabled. This value is read-only and it never changes.

Parameter range

0 - 0xFFFFFFFF [read-only]

Default

Set in the factory

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 destination address that the device uses for transmissions in Transparent mode.

This destination address is also used for outgoing I/O samples in both Transparent and API modes.

0x000000000000FFFF is the broadcast address. It is also used as the polling address when the device functions as end device.

Parameter range

0 - 0xFFFFFFFF

Default

0

DL (Destination Address Low)

Set or display the lower 32 bits of the 64-bit destination address. When you combine **DH** with **DL**, it defines the destination address that the device uses for transmissions in Transparent mode. This destination address is also used for outgoing I/O samples in both Transparent and API modes.

0x000000000000FFFF is the broadcast address. It is also used as the polling address when the device functions as end device.

Parameter range

0 - 0xFFFFFFFF

Default

0xFFFF (broadcast)

RR (Unicast Mac Retries)

Set or read the maximum number of MAC level packet delivery attempts for unicasts. If **RR** is non-zero, the sent unicast packets request an acknowledgment from the recipient. Unicast packets can be retransmitted up to **RR** times if the transmitting device does not receive a successful acknowledgment.

Parameter range

0 - 0xF

Default

0xA (10 retries)

MT (Broadcast Multi-Transmits)

Set or read the number of additional MAC-level broadcast transmissions. All broadcast packets are transmitted **MT+1** times to increase chances that they are received.

Parameter range

0 - 0xF

Default

3

TO (Transmit Options)

The device's transmit options. The device uses these options for all transmissions. API transmissions can override this using the TxOptions field in the API frame.

Parameter range

0x40-0xDF

Bit field:

Bit	Meaning	Description
0	Disable ACK	Disable acknowledgments on all unicasts
1	Disable RD	Disable Route Discovery on all DigiMesh unicasts
2	NACK	Enable a NACK messages on all DigiMesh API packets
3	Trace Route	Enable a Trace Route on all DigiMesh API packets
4	Reserved	<set this bit to 0>

Bit	Meaning	Description
5	Reserved	<set this bit to 0>
6,7	Delivery method	b'00 = <invalid option> b'01 = Point-multipoint (0x40) b'10 = Directed Broadcast (0x80) b'11 = DigiMesh (0xC0)

Default

0xC0

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 frame - 0x10](#) in API mode. In API mode, transmissions performed with the [Explicit Addressing Command frame - 0x11](#) ignore this parameter.

If you set this value to **0x12** (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)

DigiMesh configuration commands

The following commands affect outgoing transmissions in a DigiMesh network.

MR (Mesh Unicast Retries)

Set or read the maximum number of network packet delivery attempts. If **MR** is non-zero, the packets a device sends request a network acknowledgment, and can be resent up to **MR+1** times if the device does not receive an acknowledgment.

Changing this value dramatically changes how long a route request takes.

We recommend that you set this value to **1**.

If you set this parameter to **0**, it disables network ACKs. Initially, the device can find routes, but a route will never be repaired if it fails.

Parameter range

0 - 7 mesh unicast retries

Default

1

BH (Broadcast Hops)

The maximum transmission hops for broadcast data transmissions.

If you set **BH** greater than **NH (Network Hops)**, the device uses the value of **NH**.

If you set **BH** to **0**, the device uses **NH** as a limit to the maximum number of hops.

When working in API mode, the **Broadcast Radius** field in the API frame is used instead of this configuration.

Parameter range

0 - 0x20

Default

0

NH (Network Hops)

Sets or displays the maximum number of hops across the network. This parameter limits the number of hops for both unicasts and broadcasts. For example a RREQ is discarded after **NH** hops occur, preventing the route to a node more than **NH** hops away from being created. Without a route, unicasts will not work to that node. You can use this parameter to calculate the maximum network traversal time.

You must set this parameter to the same value on all nodes in the network.

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

1 - 0x20 (1 - 32 hops)

Default

7

NN (Network Delay Slots)

Set or read the maximum random number of network delay slots before rebroadcasting a network packet.

One network delay slot is approximately 13 ms.

Parameter range

1 - 0xA network delay slots

Default

3

DM (DigiMesh Options)

A bit field mask that you can use to enable or disable DigiMesh features. We highly recommend that you set the same **DM** value on every node on the network, otherwise you may encounter unexpected behavior when attempting to use the DigiMesh diagnostic features.

Bit:

0: Disable aggregator updates. When set to 1, the device does not issue or respond to **AG** requests.

1: Disable Trace Route and NACK responses. When set to 1, the device does not generate or respond to Trace Route or NACK requests.

Parameter range

0 - 0x03 (bit field)

Default

0

AG (Aggregator Support)

The **AG** command sends a broadcast through the network that has the following effects on nodes that receive the broadcast:

- The receiving node establishes a DigiMesh route back to the originating node, if there is space in the routing table.
- The **DH** and **DL** of the receiving node update to the address of the originating node if the **AG** parameter matches the current **DH/DL** of the receiving node.
- API-enabled devices with updated **DH** and **DL** send an Aggregate Addressing Update frame (0x8E) out the serial port.

Parameter range

Any 64-bit address

Default

N/A

Diagnostic commands - addressing timeouts

The following AT commands provide the transmission and discovery timeout values.

%H (MAC Unicast One Hop Time)

The MAC unicast one hop time timeout in milliseconds. If you change the MAC parameters it can change this value.

The time to send a unicast between two nodes in the network should not exceed the product of the unicast one hop time (**%H**) and the number of hops between those two nodes.

Parameter range

[read-only]

Default

N/A

%8 (MAC Broadcast One Hop Time)

The MAC broadcast one hop time timeout in milliseconds. If you change MAC parameters, it can change this value.

The time to send a broadcast between two nodes in the network should not exceed the product of the broadcast one hop time (**%8**) and the number of hops between those two nodes.

Parameter range

[read-only]

Default

N/A

N? (Network Discovery Timeout)

The maximum response time, in milliseconds, for **ND** (Network Discovery) responses and **DN** (Discover Node) responses. The timeout is the sum of **NT** (Network Discovery Back-off Time) and the network propagation time.

Parameter range

This is a read-only parameter, however, its value increases or decreases as **NT** increases or decreases and you can modify **NT**.

Default

N/A

Security commands

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

EE (Encryption Enable)

Enables or disables Advanced Encryption Standard (AES) encryption. See bit 2 of [C8 \(Compatibility Options\)](#), which controls the encryption mode.

Set this command parameter the same on all devices in a network.

Parameter range

0 - 1

Parameter	Description
0	Encryption Disabled
1	Encryption Enabled

Default

0

KY (AES Encryption Key)

The Link Key used for encryption and decryption. If [C8 \(Compatibility Options\)](#) bit 2 is cleared, encryption/decryption uses the 256 bits of the **KY** value (all 64 ASCII characters of the **KY** value). **C8** bit 2 sets encryption/decryption, and uses the last 32 ASCII characters of the 256-bit **KY** value entered.

This command is write-only and cannot be read. If you attempt to read **KY**, the device returns an **OK** status.

Set this command parameter the same on all devices in a network.

Parameter range

256-bit value (up to 32 hex bytes/64 ASCII bytes)

Default

0

RF interfacing commands

The following AT commands affect the RF interface of the device.

PL (TX Power Level)

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

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

PL setting	XBee3 TX power	XBee3-PRO TX power
4	8 dBm	19 dBm
3	5 dBm	15 dBm
2	2 dBm	8 dBm
1	-1 dBm	3 dBm
0	-5 dBm	-5 dBm

Default

4

PP (Output Power in dBm)

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

-5 dBm = 0xFB.

Parameter range

0 - 0xFF [read-only]

Default

N/A

CA (CCA Threshold)

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

The **CA** parameter is measured in units of -dBm.

Parameter range

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

Default

0x0 (CCA disabled)

DB (Last Packet RSSI)

Reports the RSSI in -dBm of the last received RF data packet. **DB** returns a hexadecimal value for the -dBm measurement.

For example, if **DB** returns 0x60, then the RSSI of the last packet received was -96 dBm.

DB only indicates the signal strength of the last hop. It does not provide an accurate quality measurement for a multihop link.

If the XBee3 DigiMesh RF Module has been reset and has not yet received a packet, **DB** reports **0**.

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

Parameter range

0 - 0xFF [read-only]

Default

0

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

EC (CCA Failures)

Sets or displays the number of frames that were blocked and not sent due to CCA failures or receptions in progress. If CCA is disabled (**CA** is **0**), then this count only increments for frames that are blocked due to receive in progress. When this count reaches its maximum value of **0xFFFF**, it stops counting.

You can reset **EC** to **0** (or any other value) at any time to make it easier to track errors. This value is volatile (the value does not persist in the device's memory after a power-up sequence).

Parameter range

0 - 0xFFFF

Default

0x0

BC (Bytes Transmitted)

The number of RF bytes transmitted. The firmware counts every byte of every packet, including MAC/PHY headers and trailers.

You can reset the counter to any 32-bit value by appending 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 - 0xFFFFFFFF

Default

N/A (0 after reset)

GD (Good Packets Received)

This count increments when a device receives a good frame with a valid MAC header on the RF interface. Received MAC ACK packets do not increment this counter. Once the number reaches 0xFFFF, it 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

N/A (0 after reset)

TR (Transmission Failure Count)

This count increments whenever a MAC transmission attempt exhausts all MAC retries without ever receiving a MAC acknowledgment message from the destination node. Once the number reaches 0xFFFF, it does not count further events.

To reset the counter to any 16-bit value, append a hexadecimal parameter to the command.

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

Parameter range

0 - 0xFFFF

Default

N/A (0 after reset)

UA (Unicasts Attempted Count)

The number of unicast transmissions expecting an acknowledgment (when **RR** > 0).

To reset the counter to any 16-bit value, append a hexadecimal parameter to the command.

UA is a volatile value—that is, the value does not persist across device resets.

Parameter range

0 - 0xFFFF

Default

0

ED (Energy Detect)

Starts an energy detect scan. This command accepts an argument to specify the time in milliseconds to scan all channels. The device loops through all the available channels until the time elapses. It returns the maximal energy on each channel, a comma follows each value, and the list ends with a carriage return. The values returned reflect the energy level that ED detects in -dBm units.

Parameter range

0 - 0x80

Default

0xA (10 ms)

Sleep settings commands

The following commands enable and configure the low power sleep modes of the device.

SM command

Sets or displays the sleep mode of the device.

Normal mode is always awake. Pin sleep modes allow you to wake the device with the SLEEP_REQUEST line. Asynchronous cyclic mode sleeps for **SP** time and briefly wakes, checking for activity. The XBee3 DigiMesh RF Module does not support synchronous sleep.

Parameter range

0 - 5

Parameter	Description
0	No sleep (disabled)
1	Pin sleep
2	Reserved
3	Reserved
4	Cyclic Sleep Remote
5	Cyclic Sleep Remote with pin wakeup

Default

0

SP (Sleep Time)

Sets or displays the device's sleep time. This command defines the amount of time the device sleeps per cycle.

For a node operating as an Indirect Messaging Coordinator, this command defines the amount of time that it will hold an indirect message for an end device. The coordinator will hold the message for (2.5 * **SP**).

Parameter range

0x1 - 0x15F900 (x 10 ms) (4 hours)

Default

0xC8

ST (Wake Time)

Sets or displays the wake time of the device.

For devices in cyclic sleep, **ST** defines the amount of time that a device stays awake after it receives RF or serial data.

Parameter range

0x1 - 0x36EE80 (x 1 ms)

Default

0x7D0 (2 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 of Asynchronous Cyclic Sleep. This allows external circuitry to sleep longer than the **SP** time.

Parameter range

1 - 0xFFFF

Default

1

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

SO (Sleep Options)

Set or read the sleep options bit field of a device. This command is a bitmask. You can set or clear any of the available sleep option bits.

Parameter range

0x0000, 0x0100

For asynchronous sleep devices, the following sleep bit field options are defined:

Bit	Option
8	Always wake for ST time

Default

0

UART interface commands

The following AT commands are serial interfacing commands.

BD (Baud Rate)

To request non-standard baud rates with values above 0x80, you can use the Serial Console toolbar in XCTU to configure the serial connection (if the console is connected), or click the **Connect** button (if the console is not yet connected).

When you send non-standard baud rates to a device, it stores the closest interface data rate represented by the number in the **BD** register. Read the **BD** command by sending **ATBD** without a parameter value, and the device returns the value stored in the **BD** register.

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

Parameter	Description
0x4	19200 b/s
0x5	38400 b/s
0x6	57600 b/s
0x7	115200 b/s
0x8	230400 b/s
0x9	460,800 b/s
0xA	921,600 b/s
0x4B0 (1200 b/s) to 0xEC400 (967680 b/s) (non standard baud rates)	

Default

0x03 (9600 baud)

NB (Parity)

Set or read the serial parity settings for UART communications.

Parameter range

0x00 - 0x02

Parameter	Description
0x00	No parity
0x01	Even parity
0x02	Odd parity

Default

0

SB (Stop Bits)

Sets or displays the number of stop bits for UART communications.

Parameter range

0 - 1

Parameter	Configuration
0	One stop bit
1	Two stop bits

Default

0

FT (Flow Control Threshold)

Set or display the flow control threshold.

The device de-asserts CTS when FT bytes are in the UART receive buffer. It re-asserts CTS when less than FT bytes are in the UART receive buffer.

Parameter range

0x07 - 0x66 bytes

Default

0x51

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

AP (API Enable)

Set or read the API mode setting. The device can format the RF packets it receives into API frames and sends them out the serial port.

When you enable API, you must format the serial data as API frames because Transparent operating mode is disabled.

Parameter range

0 - 2

Parameter	Description
0	API disabled (operate in Transparent mode)
1	API enabled
2	API enabled (with escaped control characters)

Default

0

AO (API Options)

The API data frame output format for RF packets received.

Use **AO** to enable different API output frames.

Parameter range

0 - 1

Parameter	Description
0	API Rx Indicator - 0x90, this is for standard data frames.
1	API Explicit Rx Indicator - 0x91, this is for Explicit Addressing data frames.

Default

0

AZ (Extended API Options)

Optionally output additional ZCL messages that would normally be masked by the XBee application. Use this when debugging OTA firmware updates by enabling client-side messages to be sent out of the serial port.

Parameter range

0 - 2

Parameter	Description
0	Suppress ZCL output
1	Reserved
2	Output supported ZCL packets

Default

0

Command mode options

The following commands affect how [Command mode](#) operates.

CC (Command Character)

The character value the device uses to enter Command mode.

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 a device does not receive any valid commands within this time period, it returns to Transparent mode or API mode.

Parameter range

2 - 0x1770 (x 100 ms)

Default

0x64 (10 seconds)

GT (Guard Time)

Set the required period of silence before and after the command sequence characters of the Command mode sequence, **GT + CC + GT** (including spaces). The period of silence prevents inadvertently entering Command mode. For more information, see [Enter Command mode](#).

Parameter range

0x2 - 0x6D3 (x 1 ms)

Default

0x3E8 (one second)

CN (Exit Command mode)

Executable command. **CN** immediately exits Command mode and applies pending changes.

Parameter range

N/A

Default

N/A

UART pin configuration commands

The following commands are related to pin configuration for the UART interface.

D6 (DIO6/RTS Configuration)

Sets or displays the DIO6/RTS configuration (Micro pin 27/SMT pin 29/TH pin 16).

Parameter range

0, 1, 3 - 5

Parameter	Description
0	Disabled
1	$\overline{\text{RTS}}$ flow control
2	N/A
3	Digital input
4	Digital output, low
5	Digital output, high

Default

0

D7 (DIO7/CTS Configuration)

Sets or displays the DIO7/ $\overline{\text{CTS}}$ configuration (Micro pin 24/SMT pin 25/TH pin 12).

Parameter range

0, 1, 3 - 7

Parameter	Description
0	Disabled
1	$\overline{\text{CTS}}$ flow control
2	N/A
3	Digital input
4	Digital output, low
5	Digital output, high
6	RS-485 enable, low
7	RS-485 enable, high

Default

1

P3 (DIO13/UART_DOUT)

Sets or displays the DIO13/UART_DOUT configuration (Micro pin 3/SMT pin 3/TH pin 2).

Parameter range

0, 1, 3 - 5

Parameter	Description
0	Disabled
1	UART DOUT
2	N/A
3	Digital input
4	Digital output, low
5	Digital output, high

Default

1

P4 (DIO14/UART_DIN Configuration)

Sets or displays the DIO14/UART_DIN configuration (Micro pin 4/SMT pin 4/TH pin 3).

Parameter range

0, 1, 3 - 5

Parameter	Description
0	Disabled
1	UART DIN
2	N/A
3	Digital input
4	Digital output, low
5	Digital output, high

Default

1

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 XBee3; see **D1** through **D4** and **P2** for through-hole SPI support.

P5 (DIO15/SPI_MISO Configuration)

Sets or displays the DIO15/SPI_MISO configuration (Micro pin 16/SMT pin 17). This only applies to surface-mount and micro devices.

Parameter range

0, 1, 4, 5

Parameter	Description
0	Disabled
1	SPI_MISO
2	N/A
3	N/A
4	Digital output, low
5	Digital output, high

Default

1

P6 (DIO16/SPI_MOSI Configuration)

Sets or displays the DIO16/SPI_MOSI configuration (Micro pin 15/SMT pin 16). This only applies to surface-mount and micro devices.

Parameter range

0, 1, 4, 5

Parameter	Description
0	Disabled
1	SPI_MOSI
2	N/A
3	N/A
4	Digital output, low
5	Digital output, high

Default

1

P7 (DIO17/SPI_SSEL Configuration)

Sets or displays the DIO17/SPI_SSEL configuration (Micro pin 14/SMT pin 15). This only applies to surface-mount and micro devices.

Parameter range

0 - 1, 4, 5

Parameter	Description
0	Disabled
1	SPI_SSEL
2	N/A
3	N/A
4	Digital output, low
5	Digital output, high

Default

1

P8 (DIO18/SPI_CLK Configuration)

Sets or displays the DIO18/SPI_CLK configuration (Micro pin 13/SMT pin 14). This only applies to surface-mount and micro devices.

Parameter range

0, 1, 4, 5

Parameter	Description
0	Disabled
1	SPI_CLK
2	N/A
3	N/A
4	Digital output, low
5	Digital output, high

Default

1

P9 (DIO19/SPI_ATTN Configuration)

Sets or displays the DIO19/SPI_ATTN configuration (Micro pin 11/SMT pin 12). This only applies to surface-mount and micro devices.

Parameter range

0, 1, 4, 5

Parameter	Description
0	Disabled
1	SPI_ <u>ATTN</u>
2	N/A
3	N/A
4	Digital output, low
5	Digital output, high

Default

1

I/O settings commands

The following commands configure the various I/O lines available on the XBee3 DigiMesh RF Module.

DO (DIO0/ADC0/Commissioning Configuration)

Sets or displays the DIO0/ADC0/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

D1 (DIO1/ADC1/TH_SPI_ATTN Configuration)

Sets or displays the DIO1/ADC1/TH_SPI_ATTN configuration (Micro pin 30/SMT pin 32/TH pin 19).

Parameter range

SMT/MMT: 0, 2 - 5

TH: 0 - 5

Parameter	Description
0	Disabled
1	SPI_ATT \overline{N} 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/ADC2/TH_SPI_CLK Configuration)

Sets or displays the DIO2/ADC2/TH_SPI_CLK configuration (Micro pin 29/SMT pin 31/TH pin 18).

Parameter range

SMT/MMT: 0, 2 - 5

TH: 0 - 5

Parameter	Description
0	Disabled
1	SPI_CLK for through-hole devices N/A for surface-mount devices
2	ADC
3	Digital input
4	Digital output, low
5	Digital output, high

Default

0

D3 (DIO3/ADC3/TH_SPI_SSEL Configuration)

Sets or displays the DIO3/ADC3/TH_SPI_SSEL configuration (Micro pin 28/SMT pin 30/TH pin 17).

Parameter range

SMT/MMT: 0, 2 - 5

TH: 0 - 5

Parameter	Description
0	Disabled
1	SPI_SSEL for the through-hole device N/A for surface-mount device
2	ADC
3	Digital input
4	Digital output, low
5	Digital output, high

Default

0

D4 (DIO4/TH_SPI_MOSI Configuration)

Sets or displays the DIO4/TH_SPI_MOSI configuration (Micro pin 23/SMT pin 24/TH pin 11).

Parameter range

SMT/MMT: 0, 3 - 5

TH: 0, 1, 3 - 5

Parameter	Description
0	Disabled
1	SPI_MOSI for the through-hole device N/A for the surface-mount and micro device
2	N/A
3	Digital input
4	Digital output, low
5	Digital output, high

Default

0

D5 (DIO5/Associate Configuration)

Sets or displays the DIO5/ASSOCIATED_INDICATOR configuration (Micro pin 26/SMT pin 28/TH pin 15).

Parameter range

0, 1, 3 - 5

Parameter	Description
0	Disabled
1	Associate LED indicator - blinks when associated
2	N/A
3	Digital input
4	Digital output, low
5	Digital output, high

Default

1

D8 (DIO8/DTR/SLP_Request Configuration)

Sets or displays the DIO8/ $\overline{\text{DTR}}$ /SLP_RQ configuration (Micro pin 9/SMT pin 10/TH pin 9).

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)
2	N/A
3	Digital input
4	Digital output, low
5	Digital output, high

Default

1

D9 (DIO9/ON_SLEEP Configuration)

Sets or displays the DIO9/ON_ $\overline{\text{SLEEP}}$ configuration (Micro pin 25/SMT pin 26/TH pin 13).

Parameter range

0, 1, 3 - 5

Parameter	Description
0	Disabled
1	ON/ $\overline{\text{SLEEP}}$ indicator

Parameter	Description
2	N/A
3	Digital input
4	Digital output, low
5	Digital output, high

Default

1

P0 (DIO10/RSSI/PWM0 Configuration)

Sets or displays the DIO10/RSSI/PWM0 configuration (Micro pin 7/SMT pin 7/TH pin 6).

When configured as RSSI PWM output, the device outputs a PWM signal with a duty cycle equivalent to the dBm of the received packet.

Use [RP command](#) to configure the timeout.

When configured as PWM output (**2**): you can use **MO** to explicitly control the PWM0 output. When used with [Analog line passing](#), PWM0 corresponds with ADC0.

Parameter range

0 - 5

Parameter	Description
0	Disabled
1	RSSI PWM output
2	PWM0 output. MO (PWM0 Duty Cycle) or I/O line passing control the value.
3	Digital input
4	Digital output, low
5	Digital output, high

Default

1

P1 (DIO11/PWM1 Configuration)

Sets or displays the DIO11 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) or I/O line passing control the value.
3	Digital input
4	Digital output, low
5	Digital output, high

Default

0

P2 (DIO12/TH_SPI_MISO Configuration)

Sets or displays the DIO12/TH_SPI_MISO configuration (Micro pin 5/SMT pin 5/TH pin 4).

Parameter range

SMT/MMT: 0, 3 - 5

TH: 0, 1, 3 - 5

Parameter	Description
0	Disabled
1	SPI_MISO for the through-hole device N/A for the surface-mount and micro device
2	N/A
3	Digital input
4	Digital output, low
5	Digital output, high

Default

0

PR (Pull-up/Down Resistor Enable)

The bit field that configures the internal pull-up 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
17	DIO17	14	15	N/A
18	DIO18	13	14	N/A
19	DIO19	11	12	N/A

Parameter range

0 - 0xFFFFF (bit field)

Default

Through-hole: 0 - 0xFFFF

SMT/MMT: 0 - 0xFFFFF

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.

See [PR \(Pull-up/Down Resistor Enable\)](#) for the bit mappings.

Parameter range

Through-hole: 0 - 0xFFFF

SMT/MMT: 0 - 0xFFFFF

Default

0xFFFF

IO (Set Digital I/O Lines)

Sets digital output levels. This allows DIO lines setup as outputs to be changed through Command mode.

Parameter range

8-bit bit map; each bit represents the level of an I/O line set up as an output

Default

N/A

M0 (PWM0 Duty Cycle)

The duty cycle of the PWM0 line (Micro pin 7/SMT pin 7).

If [IA \(I/O Input Address\)](#) is set correctly and [P0 \(DIO10/RSSI/PWM0 Configuration\)](#) is configured as PWM0 output, incoming AD0 samples automatically modify the PWM0 value. See [PT \(PWM Output Timeout\)](#).

To configure the duty cycle of PWM0:

1. Enable PWM0 output (**P0** = 2).
2. Change **M0** to the desired value.
3. Apply settings (use **CN** or **AC**).

The PWM period is 64 μ s and there are 0x03FF (1023 decimal) steps within this period. When **M0** = 0 (0% PWM), 0x01FF (50% PWM), 0x03FF (100% PWM), and so forth.

Parameter range

0 - 0x3FF

Default

0

M1 (PWM1 Duty Cycle)

If [IA \(I/O Input Address\)](#) is set correctly and [P1 \(DIO11/PWM1 Configuration\)](#) is configured as PWM1 output, incoming AD0 samples automatically modify the PWM1 value. See [PT \(PWM Output Timeout\)](#).

To configure the duty cycle of PWM1:

1. Enable PWM1 output (**P1** = 2).
2. Change **M1** to the desired value.
3. Apply settings (use **CN** or **AC**).

The PWM period is 64 μ s 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 command

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 command

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 of 250 ms.

For all other **LT** values, the firmware measures **LT** in 10 ms increments.

Parameter range

0, 0x14 - 0xFF (x 10 ms)

Default

0

CB (Commissioning Button)

Use **CB** to simulate Commissioning Pushbutton presses in software.

You can enable a physical commissioning pushbutton with [D0 \(DIO0/ADC0/Commissioning Configuration\)](#).

Set the parameter value to the number of button presses that you want to simulate. For example, send **CB1** to perform the action of pressing the Commissioning Pushbutton once.

Parameter range

1, 4

Parameter	Description
1	Keeps device awake for 30 seconds.
4	Restore defaults (equivalent to sending an RE (Restore Defaults)).

Default

N/A

I/O sampling commands

The following commands configure I/O sampling on an originating device. Any I/O sample generated by this device is sent to the address specified by **DH** and **DL**. You must configure at least one I/O line as an input or output for a sample to be generated.

IS (I/O Sample)

Immediately forces an I/O sample to be generated. 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 [AT Command Response frame - 0x88](#).

If the device receives ERROR as a response to an **IS** query, there are no valid I/O lines to sample.

Parameter range

N/A

Default

N/A

IR (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 (DIO Change Detect)

Set or read the digital I/O pins to monitor for changes in the I/O state.

IC works with the individual pin configuration commands (**D0 - D9, P0 - P4**). If the device detects a change on an enabled digital I/O pin, it immediately transmits a digital I/O sample to the address specified by **DH + DL**. If sleep is enabled, the edge transition must occur during a wake period to trigger a change detect.

The data transmission contains only DIO data.

IC is a bitmask you can use to enable or disable edge detection on individual digital I/O lines. Only DIO0 through DIO14 can be sampled using a Change Detect.

Set unused bits to **0**.

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
12	DIO12	5	5	4
13	N/A	N/A	N/A	N/A
14	N/A	N/A	N/A	N/A
15	N/A	N/A	N/A	N/A

Parameter range

0 - 0x7FFF

Default

0

AV (Analog Voltage Reference)

The analog voltage reference used for A/D sampling.

ADC lines are 10-bit analog inputs.

Parameter range

0 - 2

Parameter	Description
0	1.25 V reference
1	2.5 V reference
2	VDD reference

Default

0

IF (Sleep Sample Rate)

Set or read the number of sleep cycles that must elapse between periodic I/O samples. This allows the firmware to take I/O samples only during some wake cycles. During those cycles, the firmware takes I/O samples at the rate specified by [IR \(Sample Rate\)](#).

To enable periodic sampling, set **IR** to a non-zero value, and enable the analog or digital I/O functionality of at least one device pin. The sample rate is measured in milliseconds.

For more information, see the following commands:

- [D0 \(DIO0/ADC0/Commissioning Configuration\)](#) through [D9 \(DIO9/ON_SLEEP Configuration\)](#)
- [P0 \(DIO10/RSSI/PWM0 Configuration\)](#) through [P2 \(DIO12/TH_SPI_MISO Configuration\)](#)

Parameter range

1 - 0xFFFF (x 1 ms)

Default

1

I/O line passing commands

Configure the device for I/O line passing. When enabled, incoming I/O sample data will affect the state of analog and digital lines that are configured as output.

IA (I/O Input Address)

The source address of the device to which outputs are bound.

To disable I/O line passing, set all bytes to **0xFF**.

To allow any I/O packet addressed to this device (including broadcasts) to change the outputs, set **IA** to **0xFFFFF**.

Parameter range

0 - 0xFFFF FFFF FFFF FFFF

Default

0xFFFFFFFFFFFFFFFF (I/O line passing disabled)

IU (Send I/O Sample to Serial Port)

Indicates whether or not I/O samples should be sent to the serial port. 0 suppresses output; 1 allows output (only if the device is in API mode).

Parameter range

0 - 1

Parameter	Description
0	Disabled
1	Enabled

Default

1

T0 (D0 Timeout)

Specifies how long pin **D0** ([DIO0/ADC0/Commissioning Configuration](#)) holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0x1770 (x 100 ms)

Default

0

T1 (D1 Output Timeout)

Specifies how long pin **D1** ([DIO1/ADC1/TH_SPI_ATTN Configuration](#)) holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0x1770 (x 100 ms)

Default

0

T2 (D2 Output Timeout)

Specifies how long pin **D2** ([DIO2/ADC2/TH_SPI_CLK Configuration](#)) holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0x1770 (x 100 ms)

Default

0

T3 (D3 Output Timeout)

Specifies how long pin **D3** ([DIO3/ADC3/TH_SPI_SSEL Configuration](#)) holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0x1770 (x 100 ms)

Default

0

T4 (D4 Output Timeout)

Specifies how long pin [D4 \(DIO4/TH_SPI_MOSI Configuration\)](#) holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0x1770 (x 100 ms)

Default

0

T5 (D5 Output Timeout)

Specifies how long pin [D5 \(DIO5/Associate Configuration\)](#) holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0x1770 (x 100 ms)

Default

0

T6 (D6 Output Timeout)

Specifies how long pin [D6 \(DIO6/RTS Configuration\)](#) holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0x1770 (x 100 ms)

Default

0

T7 (D7 Output Timeout)

Specifies how long pin [D7 \(DIO7/CTS Configuration\)](#) holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0x1770 (x 100 ms)

Default

0

T8 (D8 Timeout)

Specifies how long pin [D8 \(DIO8/DTR/SLP_Request Configuration\)](#) holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0x1770 (x 100 ms)

Default

0

T9 (D9 Timeout)

Specifies how long pin **D9** ([DIO9/ON_SLEEP Configuration](#)) holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0x1770 (x 100 ms)

Default

0

Q0 (P0 Timeout)

Specifies how long pin **P0** holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0x1770 (x 100 ms)

Default

0

Q1 (P1 Timeout)

Specifies how long pin **P1** holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0x1770 (x 100 ms)

Default

0

Q2 (P2 Timeout)

Specifies how long pin **P2** holds a given value before it reverts to configured value. If set to **0**, there is no timeout.

Parameter range

0 - 0x1770 (x 100 ms)

Default

0

PT (PWM Output Timeout)

Specifies how long both PWM outputs (**P0**, **P1**) output a given PWM signal before it reverts to the configured value (**M0/M1**). If set to **0**, there is no timeout. This timeout only affects these pins when they are configured as PWM output.

Parameter range

0 - 0x1770 (x 100 ms)

Default

0xFF

Diagnostics – Firmware/Hardware Information

The following AT commands provide information about the XBee3 DigiMesh RF Module hardware and firmware.

VR (Firmware Version)

Reads the firmware version on a device.

Parameter range

0x3000 - 0x30FF [read-only]

Default

Set in the firmware

VL (Version Long)

Shows detailed version information including the application build date and time.

Parameter range

N/A

Default

N/A

VH (Bootloader Version)

Reads the bootloader version of the device.

Parameter range

N/A

Default

N/A

HV (Hardware Version)

Display the hardware version number of the device.

Parameter range

0 - 0xFFFF [read-only]

Default

Set in firmware

%C (Hardware/Software Compatibility)

Specifies what firmware is compatible with this device's hardware. **%C** is compared to the "compatibility_number" field of the firmware configuration xml file. Firmware with a compatibility number lower than the value returned by **%C** cannot be loaded onto the board. If an invalid firmware is loaded, the device will not boot until a valid firmware is reloaded.

Parameter range

[read-only]

Default

N/A

%V (Supply Voltage)

Reads the voltage on the Vcc pin in mV.

Parameter range

0 - 0xFFFF (in mV) [read only]

Default

N/A

TP (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

DD (Device Type Identifier)

Stores the Digi device type identifier value. Use this value to differentiate between multiple types of devices.

Parameter range

0 - 0xFFFFFFFF

Default

0x50000

CK (Configuration CRC)

Reads the cyclic redundancy check (CRC) of the current AT command configuration settings to determine if the configuration has changed.

After a firmware update this command may return a different value.

Parameter range

0 - 0xFFFF [read-only]

Default

N/A

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

Memory access commands

This section details the executable commands that provide memory access to the device.

AC (Apply Changes)

Immediately applies new settings without exiting Command mode.

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.

Writing parameters to non-volatile memory does not apply the changes immediately. However, since the device uses non-volatile memory to determine initial configuration following reset, the written parameters are applied following a reset.

Note Once you issue a **WR** command, do not send any additional characters to the device until after you receive the **OK** response. Use the **WR** command sparingly; the device's flash supports a limited number of write cycles.

Parameter range

N/A

Default

N/A

RE (Restore Defaults)

Restore device parameters to factory defaults.

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 frame - 0x17](#).

%F (Set Custom Default)

When %F is received, the XBee3 DigiMesh RF Module takes the next command received and applies it to both the current configuration and the custom defaults, so that when defaults are restored with [RE \(Restore Defaults\)](#) the custom value is used.

Parameter range

N/A

Default

N/A

!C (Clear Custom Defaults)

Clears all custom defaults. This command does not change the current settings, but only changes the defaults so that [RE \(Restore Defaults\)](#) restores settings to the factory values.

Parameter range

N/A

Default

N/A

R1 (Restore Factory Defaults)

Restores factory defaults, ignoring any custom defaults set using [%F \(Set Custom Default\)](#).

Parameter range

N/A

Default

N/A

Operate in API mode

API mode overview	116
Use the AP command to set the operation mode	116
API frame format	116

API mode overview

As an alternative to Transparent operating mode, you can use API operating mode. API mode provides a structured interface where data is communicated through the serial interface in organized packets and in a determined order. This enables you to establish complex communication between devices without having to define your own protocol. The API specifies how commands, command responses and device status messages are sent and received from the device using the serial interface or the SPI interface.

We may add new frame types to future versions of the firmware, so we recommend building the ability to filter out additional API frames with unknown frame types into your software interface.

Use the AP command to set the operation mode

Use [AP \(API Enable\)](#) to specify the operation mode:

AP command setting	Description
AP = 0	Transparent operating mode, UART serial line replacement with API modes disabled. This is the default option.
AP = 1	API operation.
AP = 2	API operation with escaped characters (only possible on UART).

The API data frame structure differs depending on what mode you choose.

API frame format

An API frame consists of the following:

- Start delimiter
- Length
- Frame data
- Checksum

API operation (AP parameter = 1)

This is the recommended API mode for most applications. The following table shows the data frame structure when you enable this mode:

Frame fields	Byte	Description
Start delimiter	1	0x7E
Length	2 - 3	Most Significant Byte, Least Significant Byte
Frame data	4 - number (n)	API-specific structure
Checksum	n + 1	1 byte

Any data received prior to the start delimiter is silently discarded. If the frame is not received correctly or if the checksum fails, the XBee replies with a radio status frame indicating the nature of the failure.

API operation with escaped characters (AP parameter = 2)

Setting API to 2 allows escaped control characters in the API frame. Due to its increased complexity, we only recommend this API mode in specific circumstances. API 2 may help improve reliability if the serial interface to the device is unstable or malformed frames are frequently being generated.

When operating in API 2, if an unescaped 0x7E byte is observed, it is treated as the start of a new API frame and all data received prior to this delimiter is silently discarded. For more information on using this API mode, see the [Escaped Characters and API Mode 2](#) in the Digi Knowledge base.

API escaped operating mode works similarly to API mode. The only difference is that when working in API escaped mode, the software must escape any payload bytes that match API frame specific data, such as the start-of-frame byte (0x7E). The following table shows the structure of an API frame with escaped characters:

Frame fields	Byte	Description	
Start delimiter	1	0x7E	
Length	2 - 3	Most Significant Byte, Least Significant Byte	Characters escaped if needed
Frame data	4 - n	API-specific structure	
Checksum	n + 1	1 byte	

Start delimiter field

This field indicates the beginning of a frame. It is always 0x7E. This allows the device to easily detect a new incoming frame.

Escaped characters in API frames

If operating in API mode with escaped characters (**AP** parameter = 2), when sending or receiving a serial data frame, specific data values must be escaped (flagged) so they do not interfere with the data frame sequencing. To escape an interfering data byte, insert 0x7D and follow it with the byte to be escaped (XORed with 0x20).

The following data bytes need to be escaped:

- 0x7E: start delimiter
- 0x7D: escape character
- 0x11: XON
- 0x13: XOFF

To escape a character:

1. Insert 0x7D (escape character).
2. Append it with the byte you want to escape, XORed with 0x20.

In API mode with escaped characters, the length field does not include any escape characters in the frame and the firmware calculates the checksum with non-escaped data.

Example: escape an API frame

To express the following API non-escaped frame in API operating mode with escaped characters:

Start delimiter	Length	Frame type	Frame Data								Checksum
			Data								
7E	00 0F	17	01 00 13 A2 00 40 AD 14 2E FF FE 02 4E 49 6D								

You must escape the 0x13 byte:

1. Insert a 0x7D.
2. XOR byte 0x13 with 0x20: 13 ⊕ 20 = 33

The following figure shows the resulting frame. Note that the length and checksum are the same as the non-escaped frame.

Start delimiter	Length	Frame type	Frame Data								Checksum
			Data								
7E	00 0F	17	01 00 7D 33 A2 00 40 AD 14 2E FF FE 02 4E 49 6D								

The length field has a two-byte value that specifies the number of bytes in the frame data field. It does not include the checksum field.

Length field

The length field is a two-byte value that specifies the number of bytes contained in the frame data field. It does not include the checksum field.

Frame data

This field contains the information that a device receives or will transmit. The structure of frame data depends on the purpose of the API frame:

Start delimiter	Length		Frame data								Checksum
			Frame type	Data							
1	2	3	4	5	6	7	8	9	...	n	n+1
0x7E	MSB	LSB	API frame type	Data							Single byte

- **Frame type** is the API frame type identifier. It determines the type of API frame and indicates how the Data field organizes the information.
- **Data** contains the data itself. This information and its order depend on the what type of frame that the Frame type field defines.

Multi-byte values are sent big-endian.

Calculate and verify checksums

To calculate the checksum of an API frame:

1. Add all bytes of the packet, except the start delimiter 0x7E and the length (the second and third bytes).
2. Keep only the lowest 8 bits from the result.
3. Subtract this quantity from 0xFF.

To verify the checksum of an API frame:

1. Add all bytes including the checksum; do not include the delimiter and length.
2. If the checksum is correct, the last two digits on the far right of the sum equal 0xFF.

Example

Consider the following sample data packet: **7E 00 0A 01 01 50 01 00 48 65 6C 6C 6F B8+**

Byte(s)	Description
7E	Start delimiter
00 0A	Length bytes
01	API identifier
01	API frame ID
50 01	Destination address low
00	Option byte
48 65 6C 6C 6F	Data packet
B8	Checksum

To calculate the check sum you add all bytes of the packet, excluding the frame delimiter **7E** and the length (the second and third bytes):

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

Add these hex bytes:

$$01 + 01 + 50 + 01 + 00 + 48 + 65 + 6C + 6C + 6F = 247$$

Now take the result of 0x247 and keep only the lowest 8 bits which in this example is 0xC4 (the two far right digits). Subtract 0x47 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 XBee3 DigiMesh RF Module will consider the packet invalid and will ignore the data.

To verify the check sum of an API packet add all bytes including the checksum (do not include the delimiter and length) and if correct, the last two far right digits of the sum will equal FF.

$$01 + 01 + 50 + 01 + 00 + 48 + 65 + 6C + 6C + 6F + B8 = 2FF$$

Frame descriptions

The following sections describe the API frames.

AT Command Frame - 0x08	121
AT Command - Queue Parameter Value frame - 0x09	123
Transmit Request frame - 0x10	125
Explicit Addressing Command frame - 0x11	128
Remote AT Command Request frame - 0x17	131
AT Command Response frame - 0x88	133
Modem Status frame - 0x8A	135
Transmit Status frame - 0x8B	136
Route Information Packet frame - 0x8D	138
Aggregate Addressing Update frame - 0x8E	141
Receive Packet frame - 0x90	143
Explicit Rx Indicator frame - 0x91	145
I/O Data Sample Rx Indicator frame - 0x92	148
Node Identification Indicator frame - 0x95	150
Remote Command Response frame - 0x97	154

AT Command Frame - 0x08

Description

Use this frame to query or set command parameters on the local device. This API command applies changes after running the command. You can query parameter values by sending the [AT Command Frame - 0x08](#) with no parameter value field (the two-byte AT command is immediately followed by the frame checksum). 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 [AT Command - Queue Parameter Value frame - 0x09](#) instead.

A [Transmit Status frame - 0x8B](#) response frame is populated with the parameter value that is currently set on the device.

Format

The following table provides the contents of the frame. For details on frame structure, see [API frame format](#).

Frame data fields	Offset	Description
Frame type	3	0x08
AT command	5-6	Command name: two ASCII characters that identify the AT command.
Parameter value	7-n	If present, indicates the requested parameter value to set the given register. If no characters are present, it queries the register.

Example

The following example illustrates an AT Command frame when you modify the device's **NH** parameter value.

Frame data fields	Offset	Example
Start delimiter	0	0x7E
Length	MSB 1	0x00
	LSB 2	0x04
Frame type	3	0x08
Frame ID	4	0x52
AT command	5	0x4E (N)
	6	0x48 (H)

Frame data fields	Offset	Example
Parameter value (NH2 = two network hops)	7	0x02
Checksum	8	0x0D

AT Command - Queue Parameter Value frame - 0x09

Description

This frame allows you to query or set device parameters. In contrast to the AT Command (0x08) frame, this frame sets new parameter values and does not apply them until you issue either:

- The **AT** Command (0x08) frame
- The **AC** command

When querying parameter values, the 0x09 frame behaves identically to the 0x08 frame; the response for this command is also an **AT** Command Response frame (0x88).

Format

The following table provides the contents of the frame. For details on frame structure, see [API frame format](#).

Frame data fields	Offset	Description
Frame type	3	0x09
Frame ID	4	Identifies the data frame for the host to correlate with a subsequent ACK. If set to 0 , the device does not send a response.
AT command	5-6	Command name: two ASCII characters that identify the AT command.
Parameter value	7-n	If present, indicates the requested parameter value to set the given register. If no characters are present, queries the register.

Example

The following example sends a command to change the baud rate (**BD**) to 115200 baud, but does not apply the changes immediately. The device continues to operate at the previous baud rate until you apply the changes.

Note In this example, you could send the parameter as a zero-padded 2-byte or 4-byte value.

Frame data fields	Offset	Example
Start delimiter	0	0x7E
Length	MSB 1	0x00
	LSB 2	0x05
Frame type	3	0x09

Frame data fields	Offset	Example
Frame ID	4	0x01
AT command	5	0x42 (B)
	6	0x44 (D)
Parameter value (BD7 = 115200 baud)	7	0x07
Checksum	8	0x68

Transmit Request frame - 0x10

Description

This frame causes the device to send payload data as an RF packet to a specific destination.

- 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.
- Set the reserved field to **0xFFFE**.
- Query the **NP** command to read the maximum number of payload bytes.

You can set the broadcast radius from **0** up to **NH**. If set to **0**, the value of **NH** specifies the broadcast radius (recommended). This parameter is only used for broadcast transmissions.

You can read the maximum number of payload bytes with the **NP** command.

Format

The following table provides the contents of the frame. For details on frame structure, see [API frame format](#).

Frame data fields	Offset	Description
Frame type	3	0x10
Frame ID	4	Identifies the data frame for the host to correlate with a subsequent ACK. If set to 0 , the device does not send a response.
64-bit destination address	5-12	MSB first, LSB last. Set to the 64-bit address of the destination device. Broadcast = 0x000000000000FFFF
Reserved	13-14	Set to 0xFFFE.
Broadcast radius	15	Sets the maximum number of hops a broadcast transmission can occur. If set to 0, the broadcast radius is set to the maximum hops value.
Transmit options	16	See the Transmit Options table below. Set all other bits to 0.
RF data	17-n	Up to NP bytes per packet. Sent to the destination device.

Transmit Options bit field

Bit field

Bit	Meaning	Description
0	Disable ACK	Disable acknowledgments on all unicasts
1	Disable RD	Disable Route Discovery on all DigiMesh unicasts
2	NACK	Enable unicast NACK messages on all DigiMesh API packets
3	Trace route	Enable a unicast Trace Route on all DigiMesh API packets
4	Reserved	<set this bit to 0>
5	Reserved	<set this bit to 0>
6,7	Delivery method	b'00 = <invalid option> b'01 = Point-multipoint (0x40) b'10 = Directed Broadcast (0x80) b'11 = DigiMesh (0xC0)

Example

The example shows how to send a transmission to a device if you disable escaping (**AP** = 1), with destination address 0x0013A200 400A0127, and payload “TxData0A”.

Frame data fields	Offset	Example
Start delimiter	0	0x7E
Length	MSB 1	0x00
	LSB 2	0x16
Frame type	3	0x10
Frame ID	4	0x01
64-bit destination address	MSB 5	0x00
	6	0x13
	7	0xA2
	8	0x00
	9	0x40
	10	0x0A
	11	0x01
	LSB 12	0x27
16-bit destination network address	MSB 13	0xFF
	LSB 14	0xFE
Broadcast radius	15	0x00

Frame data fields	Offset	Example
Options	16	0x00
RF data	17	0x54
	18	0x78
	19	0x44
	20	0x61
	21	0x74
	22	0x61
	23	0x30
	24	0x41
Checksum	25	0x13

If you enable escaping (**AP** = 2), the frame should look like:

0x7E 0x00 0x16 0x10 0x01 0x00 0x7D 0x33 0xA2 0x00 0x40 0x0A 0x01 0x27 0xFF 0xFE 0x00
 0x00 0x54 0x78 0x44 0x61 0x74 0x61 0x30 0x41 0x7D 0x33

The device calculates the checksum (on all non-escaped bytes) as [0xFF - (sum of all bytes from API frame type through data payload)].

Explicit Addressing Command frame - 0x11

Description

This frame is similar to Transmit Request (0x10), but it also requires you to specify the application-layer addressing fields: endpoints, cluster ID, and profile ID.

This frame causes the device to send payload data as an RF packet to a specific destination, using specific source and destination endpoints, cluster ID, and profile ID. These fields ignore the ones specified by **DE,SE** and **CI**.

- 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, otherwise set the 16-bit address field to the desired 16-bit destination.
- Set the reserved field to **0xFFFE**.

Query the **NP** command to read the maximum number of payload bytes. For more information, see [Diagnostics – Firmware/Hardware Information](#).

You can read the maximum number of payload bytes with the **NP** command.

Format

The following table provides the contents of the frame. For details on frame structure, see [API frame format](#).

Frame data fields	Offset	Description
Frame type	3	0x11
Frame ID	4	Identifies the data frame for the host to correlate with a subsequent ACK. If set to 0 , the device does not send a response.
64-bit destination Address	5-12	MSB first, LSB last. Set to the 64-bit address of the destination device. Broadcast = 0x000000000000FFFF
Reserved	13-14	Set to 0xFFFE .
Source Endpoint	15	Source Endpoint for the transmission.
Destination Endpoint	16	Destination Endpoint for the transmission.
Cluster ID	17-18	The Cluster ID that the host uses in the transmission.
Profile ID	19-20	The Profile ID that the host uses in the transmission.

Frame data fields	Offset	Description
Broadcast Radius	21	Sets the maximum number of hops a broadcast transmission can traverse. If set to 0, the transmission radius set to the network maximum hops value. If the broadcast radius exceeds the value of NH then the devices use the value of NH as the radius. Only broadcast transmissions use this parameter.
Transmission Options	22	See the Transmit Options table below. Set all other bits to 0.
Data Payload	23-n	Data that is sent to the destination device.

Transmit Options bit field

See [Bit field](#).

Example

The following example sends a data transmission to a device with:

- 64-bit address: 0x0013A200 01238400
- Source endpoint: 0xE8
- Destination endpoint: 0xE8
- Cluster ID: 0x11
- Profile ID: 0xC105
- Payload: TxData

Frame data fields	Offset	Example
Start delimiter	0	0x7E
Length	MSB 1	0x00
	LSB 2	0x1A
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	0x01
	10	0x23
	11	0x84
	LSB12	0x00
Reserved	13	0xFF
	14	0xFE
Source endpoint	15	0xE8
Destination endpoint	16	0xE8
Cluster ID	17	0x00
	18	0x11
Profile ID	19	0xC1
	20	0x05
Broadcast radius	21	0x00
Transmit options	22	0x00
Data payload	23	0x54
	24	0x78
	25	0x44
	26	0x61
	27	0x74
	28	0x61
Checksum	29	0xA6

Remote AT Command Request frame - 0x17

Description

Used to query or set device 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.

Format

The following table provides the contents of the frame. For details on frame structure, see [API frame format](#).

Frame data fields	Offset	Description
Frame type	3	0x17
Frame ID	4	Identifies the data frame for the host to correlate with a subsequent ACK. If set to 0 , the device does not send a response.
64-bit destination address	5-12	MSB first, LSB last. Set to the 64-bit address of the destination device.
Reserved	13-14	Set to 0xFFFE.
Remote command options	15	0x02 = Apply changes on remote. If you do not set this, you must send the AC command for changes to take effect. Set all other bits to 0.
AT command	16-17	Command name: two ASCII characters that identify the command.
Command parameter	18-n	If present, indicates the parameter value you request for a given register. If no characters are present, it queries the register. Numeric parameter values are given in binary format.

Example

The following example sends a remote command:

- Change the broadcast hops register on a remote device to 1 (broadcasts go to 1-hop neighbors only).
- Apply changes so the new configuration value takes effect immediately.

In this example, the 64-bit address of the remote device is 0x0013A200 40401122.

Frame data fields	Offset	Example
Start delimiter	0	0x7E
Length	MSB 1	0x00
	LSB 2	0x10
Frame type	3	0x17
Frame ID	4	0x01
64-bit destination address	MSB 5	0x00
	6	0x13
	7	0xA2
	8	0x00
	9	0x40
	10	0x40
	11	0x11
	LSB 12	0x22
Reserved	13	0xFF
	14	0xFE
Remote command options	15	0x02 (apply changes)
AT command	16	0x42 (B)
	17	0x48 (H)
Command parameter	18	0x01
Checksum	19	0xF5

AT Command Response frame - 0x88

Description

A device sends this frame in response to an AT Command (0x08 or 0x09) frame. Some commands send back multiple frames; for example, the **ND** command.

Format

The following table provides the contents of the frame. For details on frame structure, see [API frame format](#).

Frame data fields	Offset	Description
Frame type	3	0x88
Frame ID	4	Identifies the data frame for the host to correlate with a subsequent ACK. If set to 0 , the device does not send a response.
AT command	5-6	Command name: two ASCII characters that identify the command.
Command status	7	0 = OK 1 = ERROR 2 = Invalid command 3 = Invalid parameter 4 = Tx failure
Command data		The register data in binary format. If the host sets the register, the device does not return this field.

Example

If you change the **BD** parameter on a local device with a frame ID of 0x01, and the parameter is valid, the user receives the following response.

Frame data fields	Offset	Example
Start delimiter	0	0x7E
Length	MSB 1	0x00
	LSB 2	0x05
Frame type	3	0x88
Frame ID	4	0x01

Frame data fields	Offset	Example
AT command	5	0x42 (B)
	6	0x44 (D)
Command status	7	0x00
Command data		(No command data implies the parameter was set rather than queried)
Checksum	8	0xF0

Modem Status frame - 0x8A

Description

Devices send the status messages in this frame in response to specific conditions.

Format

The following table provides the contents of the frame. For details on frame structure, see [API frame format](#).

Frame data fields	Offset	Description
Frame type	3	0x8A
Status	4	0x00 = Hardware reset 0x01 = Watchdog timer reset 0x02 = End device successfully associated with a coordinator 0x03 = End device disassociated from coordinator or coordinator failed to form a new network 0x06 = End device successfully associated with a coordinator 0x0B = Network woke up 0x0C = Network went to sleep 0x0D Input voltage is too high

Example

When a device powers up, it returns the following API frame.

Frame data fields	Offset	Example
Start delimiter	0	0x7E
Length	MSB 1	0x00
LSB 2	LSB 2	0x02
Frame type	3	0x8A
Status	4	0x00
Checksum	5	0x75

Transmit Status frame - 0x8B

Description

When a Transmit Request (0x10, 0x11) completes, the device sends an 0x8B Transmit Status message out of the serial interface. This message indicates if the Transmit Request was successful or if it failed.

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](#).

Frame data fields	Offset	Description
Frame type	3	0x8B
Frame ID	4	The Frame ID of the response will be the same value that was used in the originating Tx request.
16-bit destination address	5	The 16-bit Network Address where the packet was delivered (if successful). If not successful, this address is 0xFFFF (destination address unknown).
	6	
Transmit retry count	7	The number of application transmission retries that occur.
Delivery status	8	0x00 = Success 0x01 = MAC ACK Failure 0x02 = Collision avoidance failure 0x21 = Network ACK Failure 0x25 = Route not found 0x31 = Internal resource error 0x32 = Internal error 0x74 = Data payload too large 0x75 = Indirect message unrequested
Discovery status	9	0x00 = No discovery overhead 0x02 = Route discovery

Example

In the following example, the destination device reports a successful unicast data transmission successful and a route discovery occurred. The outgoing Transmit Request that this response frame came from uses Frame ID of 0x47.

Frame Fields	Offset	Example
Start delimiter	0	0x7E
Length	MSB 1	0x00
	LSB 2	0x07
Frame type	3	0x8B
Frame ID	4	0x47
Reserved	5	0xFF
	6	0xFE
Transmit retry count	7	0x00
Delivery status	8	0x00
Discovery status	9	0x02
Checksum	10	0x2E

Route Information Packet frame - 0x8D

Description

If you enable NACK or the Trace Route option on a DigiMesh unicast transmission, a device can output this frame for the transmission.

Format

The following table provides the contents of the frame. For details on frame structure, see [API frame format](#).

Frame data fields	Offset	Description
Frame type	3	0x8D
Source event	4	0x11 = NACK 0x12 = Trace route
Length	5	The number of bytes that follow, excluding the checksum. If the length increases, new items have been added to the end of the list for future revisions.
Timestamp	6-9	System timer value on the node generating the Route Information Packet. The timestamp is in microseconds. Only use this value for relative time measurements because the time stamp count restarts approximately every hour.
ACK timeout count	10	The number of MAC ACK timeouts that occur.
TX blocked count	11	The number of times the transmission was blocked due to reception in progress.
Reserved	12	Reserved, set to 0s.
Destination address	13-20	The address of the final destination node of this network-level transmission.
Source address	21-28	Address of the source node of this network-level transmission.
Responder address	29-36	Address of the node that generates this Route Information packet after it sends (or attempts to send) the packet to the next hop (the Receiver node).
Successor address	37-44	Address of the next node after the responder in the route towards the destination, whether or not the packet arrived successfully at the successor node.

Example

The following example represents a possible Route Information Packet. A device receives the packet when it performs a trace route on a transmission from one device (serial number 0x0013A200 4052AAAA) to another (serial number 0x0013A200 4052DDDD).

This particular frame indicates that the network successfully forwards the transmission from one device (serial number 0x0013A200 4052BBBB) to another device (serial number 0x0013A200 4052CCCC).

Frame data fields	Offset	Example
Start delimiter	0	0x7E
Length	MSB 1	0x00
	LSB 2	0x2A
Frame type	3	0x8D
Source event	4	0x12
Length	5	0x27 0x2B
Timestamp	MSB 6	0x9C
	7	0x93
	8	0x81
	LSB 9	0x7F
ACK timeout count	10	0x00
TX blocked count	11	0x00
Reserved	12	0x00
Destination address	MSB 13	0x00
	14	0x13
	15	0xA2
	16	0x00
	17	0x40
	18	0x52
	19	0xAA
	LSB 20	0xAA

Frame data fields	Offset	Example
Source address	MSB 21	0x00
	22	0x13
	23	0xA2
	24	0x00
	25	0x40
	26	0x52
	27	0xDD
	LSB 28	0xDD
Responder address	MSB 29	0x00
	30	0x13
	31	0xA2
	32	0x00
	33	0x40
	34	0x52
	35	0xBB
	LSB 36	0xBB
Successor address	MSB 37	0x00
	38	0x13
	39	0xA2
	40	0x00
	41	0x40
	42	0x52
	43	0xCC
	LSB 44	0xCC
Checksum	45	0xD2

Aggregate Addressing Update frame - 0x8E

Description

The device sends out an Aggregate Addressing Update frame on the serial interface of an API-enabled node when an address update frame (generated by the **AG** command being issued on a node in the network) causes the node to update its **DH** and **DL** registers.

Format

The following table provides the contents of the frame. For details on frame structure, see [API frame format](#).

Frame data fields	Offset	Description
Frame type	3	0x8E
Format ID	4	Byte reserved to indicate the format of additional packet information which may be added in future firmware revisions. In the current firmware revision, this field returns 0x00.
New address	5-12	Address to which DH and DL are being set.
Old address	13-20	Address to which DH and DL were previously set.

Example

In the following example, a device with destination address (**DH/DL**) of 0x0013A200 4052AAAA updates its destination address to 0x0013A200 4052BBBB.

Frame data fields	Offset	Example
Start delimiter	0	0x7E
Length	MSB 1	0x00
	LSB 2	0x12
Frame type	3	0x8E
Format ID	4	0x00

Frame data fields	Offset	Example
New address	MSB 5	0x00
	6	0x13
	7	0xA2
	8	0x00
	9	0x40
	10	0x52
	11	0xBB
	LSB 12	0xBB
Old address	13	0x00
	14	0x13
	15	0xA2
	16	0x00
	17	0x40
	18	0x52
	19	0xAA
	20	0xAA
Checksum	21	0x19

Receive Packet frame - 0x90

Description

When a device configured with a standard API Rx Indicator (**AO (API Options) = 0**) receives an RF data packet, it sends it out the serial interface using this message type.

Format

The following table provides the contents of the frame. For details on frame structure, see [API frame format](#).

Frame data fields	Offset	Description	
Frame type	3	0x90	
64-bit source address	4-11	The sender's 64-bit address. MSB first, LSB last.	
Reserved	12-13	16-bit source address.	
Receive options	14	Bit	Interpretation
		0	Packet acknowledged
		1	Broadcast packet
		2 - 5	Reserved
		6 - 7	Delivery mode:
			b 00 Invalid
b 01 Point-to-multipoint			
b 10 Repeater mode			
		b 11 DigiMesh	
Received data	15 - n	The RF data the device receives.	

Example

In the following example, a device with a 64-bit address of 0x0013A200 40522BAA sends a unicast data transmission to a remote device with payload RxData. If **AO = 0** on the receiving device, it sends the following frame out its serial interface.

Frame data fields	Offset	Example
Start delimiter	0	0x7E

Frame data fields	Offset	Example
Length	MSB 1	0x00
	LSB 2	0x12
Frame type	3	0x90
64-bit source address	MSB 4	0x00
	5	0x13
	6	0xA2
	7	0x00
	8	0x40
	9	0x52
	10	0x2B
	LSB 11	0xAA
Reserved	12	0xFF
	13	0xFE
Receive options	14	0x01
Received data	15	0x52
	16	0x78
	17	0x44
	18	0x61
	19	0x74
	20	0x61
Checksum	21	0x11

Explicit Rx Indicator frame - 0x91

Description

When a device configured with explicit API Rx Indicator ([AO \(API Options\)](#) = 1) receives an RF packet, it sends it out the serial interface using this message type.

Note The values of the fields in the 0x91 frame (for example, endpoints and cluster ID) depend on the values sent by the initiator. If the initiator sends a [Transmit Request frame - 0x10](#) (which does not specify endpoints and cluster IDs), then the initiator sends the values configured in [DE command](#), [SE command](#), and [CI \(Cluster ID\)](#) instead.

The Cluster ID and endpoints must be 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](#).

Frame data fields	Offset	Description
Frame type	3	0x91
64-bit source address	4-11	MSB first, LSB last. The sender's 64-bit address.
Reserved	12-13	16-bit source address.
Source endpoint	14	Endpoint of the source that initiates transmission.
Destination endpoint	15	Endpoint of the destination where the message is addressed.
Cluster ID	16-17	The Cluster ID where the frame is addressed.
Profile ID	18-19	The Profile ID where the fame is addressed.
Receive options	20	Bit field: 0x00 = Packet acknowledged 0x01 = Packet was a broadcast packet 0x06, 0x07: b'01 = Point-Multipoint b'10 = Repeater mode (directed broadcast) b'11 = DigiMesh Ignore all other bits.
Received data	21-n	Received RF data.

Example

In the following example, a device with a 64-bit address of 0x0013A200 40522BAA sends a broadcast data transmission to a remote device with payload RxData.

If a device sends the transmission:

- With source and destination endpoints of 0xE0
- Cluster ID = 0x2211
- Profile ID = 0xC105

If **AO = 1** on the receiving device, it sends the following frame out its serial interface.

Frame data fields	Offset	Example
Start delimiter	0	0x7E
Length	MSB 1	0x00
	LSB 2	0x18
Frame type	3	0x91
64-bit source address	MSB 4	0x00
	5	0x13
	6	0xA2
	7	0x00
	8	0x40
	9	0x52
	10	0x2B
	LSB 11	0xAA
Reserved	12	0xFF
	13	0xFE
Source endpoint	14	0xE0
Destination endpoint	15	0xE0
Cluster ID	16	0x22
	17	0x11
Profile ID	18	0xC1
	19	0x05
Receive options	20	0x02

Frame data fields	Offset	Example
Received data	21	0x52
	22	0x78
	23	0x44
	24	0x61
	25	0x74
	26	0x61
Checksum	27	0x68

I/O Data Sample Rx Indicator frame - 0x92

Description

When you enable periodic I/O sampling or digital I/O change detection on a remote device, the UART of the device that receives the sample data sends this frame out.

Format

The following table provides the contents of the frame. For details on frame structure, see [API frame format](#).

Frame data fields	Offset	Description
Frame type	3	0x92
64-bit source address	4-11	The sender's 64-bit address.
Reserved	12-13	Reserved.
Receive options	14	Bit field: 0x01 = Packet acknowledged 0x02 = Packet is a broadcast packet Ignore all other bits
Number of samples	15	The number of sample sets included in the payload. Always set to 1.
Digital channel mask	16-17	Bitmask field that indicates which digital I/O lines on the remote have sampling enabled, if any.
Analog channel mask	18	Bitmask field that indicates which analog I/O lines on the remote have sampling enabled, if any.
Digital samples (if included)	19-20	If the sample set includes any digital I/O lines (Digital channel mask > 0), these two bytes contain samples for all enabled digital I/O lines. DIO lines that do not have sampling enabled return 0. Bits in these two bytes map the same as they do in the Digital channel mask field.
Analog sample	21-n	If the sample set includes any analog I/O lines (Analog channel mask > 0), each enabled analog input returns a 2-byte value indicating the A/D measurement of that input. Analog samples are ordered sequentially from ADO/DIO0 to AD3/DIO3.

Example

In the following example, the device receives an I/O sample from a device with a 64-bit serial number of 0x0013A20040522BAA.

The configuration of the transmitting device takes a digital sample of a number of digital I/O lines and an analog sample of AD1. It reads the digital lines to be 0x0014 and the analog sample value is 0x0225.

The complete example frame is:

```
7E00 1492 0013 A200 4052 2BAA FFFE 0101 001C 0200 1402 25F9
```

Frame fields	Offset	Example
Start delimiter	0	0x7E
Length	MSB 1	0x00
	LSB 2	0x14
64-bit source address	MSB 4	0x00
	5	0x13
	6	0xA2
	7	0x00
	8	0x40
	9	0x52
	10	0x2B
	LSB 11	0xAA
Reserved	MSB 12	0xFF
	LSB 13	0xFE
Receive options	14	0x01
Number of samples	15	0x01
Digital channel mask	16	0x00
	17	0x1C
Analog channel mask	18	0x02
Digital samples (if included)	19	0x00
	20	0x14
Analog sample	21	0x02
	22	0x25
Checksum	23	0xF5

Node Identification Indicator frame - 0x95

Description

A device receives this frame when:

- it transmits a node identification message to identify itself
- **AO = 0**. If **AO = 1**, then the node identification indicator is produced as a [Explicit Rx Indicator frame - 0x91](#).

The data portion of this frame is similar to a network discovery response. For more information, see [ND \(Network Discover\)](#).

Format

The following table provides the contents of the frame. For details on frame structure, see [API frame format](#).

Frame data fields	Offset	Description	
Frame type	3	0x95	
64-bit source address	4-11	MSB first, LSB last. The sender's 64-bit address.	
Reserved	12-13	Reserved	
Receive options	14	Bit	Interpretation
		0	Packet acknowledged
		1	Broadcast packet
		2 - 5	Reserved
		6 - 7	Delivery mode:
			b 00 Invalid
			b 01 Point-to-multipoint
b 10 Repeater mode			
	b 11 DigiMesh		
Reserved	15-16	Reserved	
64-bit remote address	17-24	Indicates the 64-bit address of the remote device that transmitted the Node Identification Indicator frame.	

Frame data fields	Offset	Description
NI string	25-26	Node identifier string on the remote device. The NI string is terminated with a NULL byte (0x00).
Reserved	27-28	Reserved
Device type	29	0 = Coordinator 1 = Normal Mode 2 = End Device For more options, see NO (Network Discovery Options)
Source event	30	1 = Frame sent by node identification pushbutton event - see D0 (DIO0/ADC0/Commissioning Configuration)
Digi Profile ID	31-32	Set to the Digi application profile ID
Digi Manufacturer ID	33-34	Set to the Digi Manufacturer ID
Digi DD value (optional)	35-38	Reports the DD value of the responding device. Use the NO command to enable this field.
RSSI (optional)	39	Received signal strength indicator. Use the NO command to enable this field.

Example

If you press the commissioning pushbutton on a remote device with 64-bit address 0x0013A200407402AC and a default **NI** string sends a Node Identification, all devices on the network receive the following node identification indicator:

Frame data fields	Offset	Example
Start delimiter	0	0x7E
Length	MSB 1	0x00
	LSB 2	0x25
Frame type	3	0x95

Frame data fields	Offset	Example
64-bit source address	MSB 4	0x00
	5	0x13
	6	0xA2
	7	0x00
	8	0x40
	9	0x74
	10	0x02
	LSB 11	0xAC
Reserved	12	0xFF
	13	0xFE
Receive options	14	0xC2
Reserved	15	0xFF
	16	0xFE
64-bit remote address	MSB 17	0x00
	18	0x13
	19	0xA2
	20	0x00
	21	0x40
	22	0x74
	23	0x02
	LSB 24	0xAC
NI string	25	0x20
	26	0x00
Reserved	27	0xFF
	28	0xFE
Device type	29	0x01
Source event	30	0x01
Digi Profile ID	31	0xC1
	32	0x05

Frame data fields	Offset	Example
Digi Manufacturer ID	33	0x10
	34	0x1E
Digi DD value (optional)	35	0x00
	36	0x0C
	37	0x00
	38	0x00
RSSI (optional)	39	0x2E
Checksum	40	0x33

Remote Command Response frame - 0x97

Description

If a device receives this frame in response to a Remote Command Request (0x17) frame, the device sends an AT Command Response (0x97) frame out the serial interface.

Some commands, such as the **ND** command, may send back multiple frames.

Format

The following table provides the contents of the frame. For details on frame structure, see [API frame format](#).

Frame data fields	Offset	Description
Frame type	3	0x97
Frame ID	4	This is the same value that is passed into the request.
64-bit source (remote) address	5-12	The address of the remote device returning this response.
Reserved	13-14	Reserved.
AT commands	15-16	The name of the command.
Command status	17	0 = OK 1 = ERROR 2 = Invalid Command 3 = Invalid Parameter
Command data	18-n	The value of the requested register.

Example

If a device sends a remote command to a remote device with 64-bit address 0x0013A200 40522BAA to query the **SL** command, and if the frame ID = 0x55, the response would look like the following example.

Frame data fields	Offset	Example
Start delimiter	0	0x7E
Length	MSB 1	0x00
	LSB 2	0x13
Frame type	3	0x97
Frame ID	4	0x55

Frame data fields	Offset	Example
64-bit source (remote) address	MSB 5	0x00
	6	0x13
	7	0xA2
	8	0x00
	9	0x40
	10	0x52
	11	0x2B
	LSB 12	0xAA
Reserved	13	0xFF
	14	0xFE
AT commands	15	0x53 (S)
	16	0x4C (L)
Command status	17	0x00
Command data	18	0x40
	19	0x52
	20	0x2B
	21	0xAA
Checksum	22	0xF4

OTA firmware upgrade process for DigiMesh 2.4

OTA/OTB file	157
Storage	158
ZCL OTA messaging	158
ZCL message output	159
Image Notify	159
Create the Image Notify request	160
Query Next Image request	162
Query Next Image response	164
Image Block request	166
Image Block response	168
Upgrade End request	171
Upgrade End response	173
OTA error handling	175

OTA/OTB file

The OTA firmware process uses a specific firmware file with a .ota extension. This file needs to be parsed and divided into 64-byte blocks. This file needs to be parsed and divided into 48 byte blocks for encrypted networks and 56 bytes for unencrypted networks.

The OTA header

The OTA firmware uses a specific firmware file with a .ota extension. We recommend parsing the OTA header from the OTA file first to obtain the firmware version, manufacturer code, image type and the size of the GBL file. These parameters are required to generate the rest of the OTA firmware upgrade messages. The format of the OTA header is:

Bytes	Field name	Description
4	OTA upgrade file identifier	Has to match 0x0BEEF11E in little endian. If it is not, then the OTA file is not a valid upgrade file.
2	OTA Header version	0x0001
2	OTA Header length	Length of the OTA Header.
2	OTA Header Field control	Bit mask that indicates if additional information is included in the image. (Read the Security Credential Version in this table).
2	Manufacturer Code	0x101E
2	Image Type	0x0000
4	File Version	The version of the firmware upgrade image.
2	Stack Version	This is set to 2 by default.
32	OTA Header String	Usually contains the Firmware image name followed by 0xFFs. For example, FFFFFFFFFFFFFFFFbg.10F3_42MD_3BX which is XB3_DM24-3F01.gblFFFFFFFFFFFFFFFF in little endian
4	Image Size	Contains the size of the .gbl file for the firmware.
0/1	Security Credential version	If bit 0 of the OTA Header Field Control is set to 1, this indicates the security credential version type that the client is required to have, before it will install the image (set to 2).
0/8	Upgrade File Destination	If bit 1 of the OTA Header Field Control is set to 1, this indicates that this OTA file contains security credential/certificate 577 data or other type of information that is specific to a particular device. Currently, we do not use this feature.
0/2	Hardware/Software Compatibility	If bit 2 of the OTA Header Field Control is set to 1.

The file version field contains additional hardware/software compatibility information. We recommend that if you intend to perform an OTA update, you use the OTA header extracted from the file so that you can avoid undesired behavior.

Hardware/software compatibility

The Hardware Software Compatibility number ensures that an incompatible firmware is not flashed on to the XBee3 DigiMesh RF Module. To obtain this value, query [%C \(Hardware/Software Compatibility\)](#) on the target device. You can successfully update the device to a firmware if, and only if, the value of **%C** of the image is greater than or equal to the value returned by the device when you query **%C**.

Parse the image blocks

To parse the image blocks:

1. Divide the contents of the underlying .gbl file into 48 byte blocks for encrypted networks and 56 byte blocks for unencrypted networks
2. Create Image Block Requests around the image blocks; see [Image Block request](#).

Note The .gbl file is placed at an offset of 75 bytes and so it is important to start parsing the image from that point in the file.

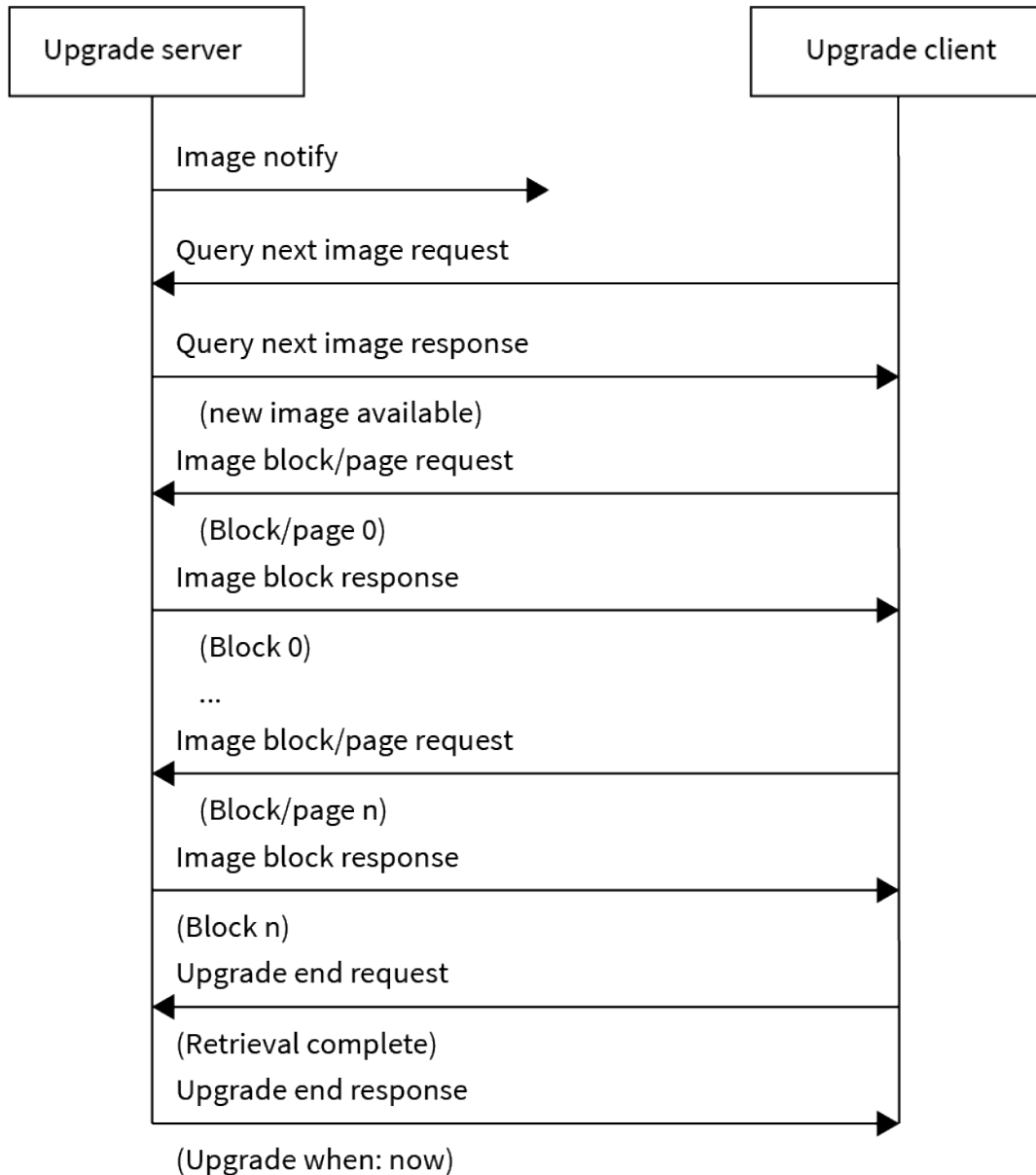
Storage

The OTA firmware image blocks are received and stored in a separate internal flash slot that is allotted exclusively for this purpose. Once all the image bytes are written to the slot, the new image must be validated by the current application before it can be used.

If the new image is deemed invalid, the running DigiMesh firmware rejects the image and continues operating with the current, valid application.

ZCL OTA messaging

The following figure provides the messaging sequence between the Server (updater node) and the Client (target node).



ZCL message output

By default ZCL messages are not printed to the UART on the client. To see these messages, set [AZ \(Extended API Options\)](#) to 2. ZCL messages received by the server are always printed to the UART.

Image Notify

The server sends the Image Notify message to the client informing the device of the presence of an update image. The Image Notify message is sent when the upgrade process is initiated from the server.

Create the Image Notify request

The Image Notify Request is an explicit transmit frame (0x11 type) passed into the server with the following information:

Frame data fields	Offset	Example	Comments
Start delimiter	0	0x7E	
Length	MSB 1	0x00	
	LSB 2	0x21	
Frame Type	3	0x11	
Frame ID	4	0x01	
64-bit destination address	MSB 5	0x00	
	6	0x13	
	7	0xA2	
	8	0xFE	
	9	0x00	
	10	0x00	
	11	0x00	
	LSB 12	0x03	
16-bit destination address	MSB 13	0x28	
	LSB 14	0x2F	
Source Endpoint	15	0xE8	
Destination Endpoint	16	0xE8	
Cluster ID	MSB 17	0x00	
	LSB 18	0x19	
Profile ID	MSB 19	0xC1	
	LSB 20	0x05	
Broadcast radius	21	0x00	
Transmit options	22	0x00	

Frame data fields			Offset	Example	Comments
Data payload	ZCL frame header	Frame control	23	0x09	
		Transaction sequence number	24	0x01	
	ZCL payload	Command ID	25	0x00	Image Notify Command ID
		Payload type	26	0x03	Contains Jitter, Image Type, Firmware Version
		Query jitter	27	0x00	
		Manufacturer ID	LSB 28	0x1E	Digi's Manufacturer ID in Little Endian
			MSB 29	0x10	
		Image type	LSB 30	0x00	Image type should be 0x0000 unless the server doesn't want to update the client
			MSB 31	0x00	
		Firmware version	LSB 32	0x01	Firmware version of the new update file in Little Endian. In this example, the version is 0x1001
			33	0x10	
			34	0x00	
			MSB 35	0x00	
		Checksum			36

Query Next Image request

The client device sends the Query Next Image request message to the server to indicate it is ready to receive a firmware image and is sent as a response to an Image Notify message. The client sends information about the existing firmware version as a part of this message. The server emits the following frame after receiving the request from the client:

Frame data fields	Offset	Example	Comments
Start delimiter	0	0x7E	
Length	MSB 1	0x00	
	LSB 2	0x1E	
Frame Type	3	0x91	
64-bit source address	MSB 4	0x00	
	5	0x13	
	6	0xA2	
	7	0xFE	
	8	0x00	
	9	0x00	
	10	0x00	
	LSB 11	0x03	
16-bit source address	MSB 12	0x28	
	LSB 13	0x2F	
Source Endpoint	14	0xE8	
Destination Endpoint	15	0xE8	
Cluster ID	MSB 16	0x00	
	LSB 17	0x19	
Profile ID	MSB 18	0xC1	
	LSB 19	0x05	
Receive options	20	0x01	

Frame data fields			Offset	Example	Comments
Data payload	ZCL frame header	Frame control	21	0x01	
		Transaction sequence number	22	0x00	
	ZCL payload	Command ID	23	0x01	Query Next Image request
		Field control	24	0x00	
		Manufacturer ID	LSB 25	0x1E	
			MSB 26	0x10	
		Image type	LSB 27	0x00	
			MSB 28	0x00	
		Firmware version	LSB 29	0x00	
			30	0x10	
			31	0x00	
			MSB 32	0x00	
Checksum			33	0x71	

Query Next Image response

The server obtains the information sent by the Client in the Query Next Image request and determines if it has a suitable image for the client. It then sends a Query Next Image response with one of the following status messages as appropriate:

- 0x00 - SUCCESS: The server is authorized to upgrade the client with the image.
- 0x98 - NO_IMAGE_AVAILABLE: The server is authorized to update the client but does not have a new OTA update image available.
- 0x7E - NOT_AUTHORIZED: The server is not authorized to update the client.

Frame data fields	Offset	Example	Comments
Start delimiter	0	0x7E	
Length	MSB 1	0x00	
	LSB 2	0x24	
Frame Type	3	0x11	
Frame ID	4	0x01	
64-bit destination address	MSB 5	0x00	
	6	0x13	
	7	0xA2	
	8	0xFE	
	9	0x00	
	10	0x00	
	11	0x00	
	LSB 12	0x03	
16-bit destination address	MSB 13	0x28	
	LSB 14	0x2F	
Source Endpoint	15	0xE8	
Destination Endpoint	16	0xE8	
Cluster ID	MSB 17	0x00	
	LSB 18	0x19	

Frame data fields			Offset	Example	Comments
Profile ID			MSB 19	0xC1	
			LSB 20	0x05	
Broadcast radius			21	0x00	
Transmit options			22	0x00	
Data payload	ZCL frame header	Frame control	23	0x09	
		Transaction sequence number	24	0x01	
	ZCL payload	Command ID	25	0x02	Query Next Image Response
		Status	26	0x00	Success = 0x00 No Image Available = 0x98 Not Authorized = 0x7E
		Manufacturer ID	LSB 27	0x1E	
			MSB 28	0x10	
		Image type	LSB 29	0x00	
			MSB 30	0x00	
		Firmware version	LSB 31	0x01	Firmware version of the new update file in Little Endian. In this example, the version is 0x1001
			32	0x10	
			33	0x00	
			MSB 34	0x00	
	Image Size	LSB 35	0x2E		
		36	0xF3		
		37	0x02		
		MSB 38	0x00		
Checksum			39	0xE5	

Image Block request

The Client generates Image Block requests to request the server for bytes of the OTA firmware image. Each image block is 64 byte long. The client also sends the file offset as a way to keep the synchronization of every block intact.

The Image Block requests are repeated by the client until all the blocks of the image are successfully obtained. The size of the OTA upgrade image is usually obtained by the client in the Query Next Image response message and hence it knows the exact number of Image Block requests it needs to send.

Frame data fields	Offset	Example	Comments
Start delimiter	0	0x7E	
Length	MSB 1	0x00	
	LSB 2	0x1E	
Frame Type	3	0x91	
64-bit source address	MSB 4	0x00	
	5	0x13	
	6	0xA2	
	7	0xFE	
	8	0x00	
	9	0x00	
	10	0x00	
	LSB 11	0x03	
16-bit source address	MSB 12	0x28	
	LSB 13	0x2F	
Source Endpoint	14	0xE8	
Destination Endpoint	15	0xE8	
Cluster ID	MSB 16	0x00	
	LSB 17	0x19	

Frame data fields			Offset	Example	Comments
Profile ID			MSB 18	0xC1	
			LSB 19	0x05	
Receive options			20	0x01	
Data payload	ZCL frame header	Frame control	21	0x01	
		Transaction sequence number	22	0x01	
Data payload	ZCL payload	Command ID	23	0x03	Image Block Request
		Field control	24	0x00	
		Manufacturer ID	LSB 25	0x1E	
			MSB 26	0x10	
		Image type	LSB 27	0x00	
			MSB 28	0x00	
		Firmware version	LSB 29	0x01	
			30	0x10	
			31	0x00	
			MSB 32	0x00	
		File Offset	LSB 33	0x00	0x0 for the first request. Offset by multiples of Image Block size. For example, 0x00000000 for the first request, 0x00000040, 0x00000080 and so on.
			34	0x00	
			35	0x00	
			LSB 36	0x00	
		Image Block Size		37	0x40
Checksum			38	0x2D	

Image Block response

The server generates an Image Block response upon receiving an Image Block request command. It responds with a SUCCESS status on being able to retrieve the data for the client. The server uses the file offset sent by the client to determine the location of the requested data within the OTA upgrade image.

If you wish to cancel the update process, send an ABORT (0x95) status.

Frame data fields	Offset	Example	Comments
Start delimiter	0	0x7E	
Length	MSB 1	0x00	
	LSB 2	0x65	
Frame Type	3	0x11	
Frame ID	4	0x01	
64-bit destination address	MSB 5	0x00	
	6	0x13	
	7	0xA2	
	8	0xFE	
	9	0x00	
	10	0x00	
	11	0x00	
	LSB 12	0x03	
16-bit destination address	MSB 13	0x28	
	LSB 14	0x2F	
Source Endpoint	15	0xE8	
Destination Endpoint	16	0xE8	
Cluster ID	MSB 17	0x00	
	LSB 18	0x19	

Frame data fields			Offset	Example	Comments
Profile ID			MSB 19	0xC1	
			LSB 20	0x05	
Broadcast radius			21	0x00	
Transmit options			22	0x00	
Data payload	ZCL frame header	Frame control	23	0x09	
Data payload		Transaction sequence number	24	0x02	
Data payload	ZCL payload	Command ID	25	0x05	Image Block Response
Data payload	ZCL payload	Status	26	0x00	Success = 0x00 Abort = 0x95
Data payload	ZCL payload	Manufacturer ID	LSB 27	0x1E	
Data payload			MSB 28	0x10	
Data payload	ZCL payload	Image type	LSB 29	0x00	
Data payload			MSB 30	0x00	
Data payload	ZCL payload	Firmware version	LSB 31	0x01	
Data payload			32	0x10	
Data payload			33	0x00	
Data payload			MSB 34	0x00	

Frame data fields			Offset	Example	Comments
Data payload	ZCL payload	File Offset	LSB 35	0x00	
Data payload	ZCL payload		36	0x00	
Data payload	ZCL payload		37	0x00	
Data payload	ZCL payload		MSB 38	0x00	
Data payload	ZCL payload	Image Block Size	39	0x40	64 byte blocks 48 byte blocks for encrypted networks 56 byte blocks for unencrypted networks
Data payload	ZCL payload	Image Block Data	40- 104	0xEB- 0x00	An image block of the size mentioned in Image Block Size
Checksum			106	0x4E	

Upgrade End request

The Upgrade End request is generated by the client after it verifies the received firmware image to ensure its integrity and validity. If the image fails any integrity checks, the client sends an Upgrade End request command to the upgrade server with INVALID_IMAGE as the status. If the image passes all integrity checks, the client sends an Upgrade End request command to the upgrade server with SUCCESS as the status.

Frame data fields	Offset	Example	Comments
Start delimiter	0	0x7E	
Length	MSB 1	0x00	
	LSB 2	0x1E	
Frame Type	3	0x91	
64-bit source address	MSB 4	0x00	
	5	0x13	
	6	0xA2	
	7	0xFE	
	8	0x00	
	9	0x00	
	10	0x00	
	LSB 11	0x03	
16-bit source address	MSB 12	0x28	
	LSB 13	0x2F	
Source Endpoint	14	0xE8	
Destination Endpoint	15	0xE8	
Cluster ID	MSB 16	0x00	
	LSB 17	0x19	
Profile ID	MSB 18	0xC1	
	LSB 19	0x05	

Frame data fields			Offset	Example	Comments
Receive options			20	0x01	
Data payload	ZCL frame header	Frame control	21	0x01	
		Transaction sequence number	22	0x30	
	ZCL payload	Command ID	23	0x06	Upgrade End Request
		Status	24	0x00	Success = 0x00 Invalid Image = 0x96 Abort = 0x95 Require More Image = 0x99
		Manufacturer ID	LSB 25	0x1E	
			MSB 26	0x10	
		Image type	LSB 27	0x00	
			MSB 28	0x00	
		Firmware version	LSB 29	0x01	
			30	0x10	
			31	0x00	
			MSB 32	0x00	
	Checksum			38	0x3B

Upgrade End response

If the server receives an Upgrade End request with a SUCCESS status, it generates an Upgrade End response along with the time at which the device should upgrade to the new image.

Frame data fields	Offset	Example	Comments
Start delimiter	0	0x7E	
Length	MSB 1	0x00	
	LSB 2	0x24	
Frame Type	3	0x11	
Frame ID	4	0x01	
64-bit destination address	MSB 5	0x00	
	6	0x13	
	7	0xA2	
	8	0xFE	
	9	0x00	
	10	0x00	
	11	0x00	
	LSB 12	0x03	
16-bit destination address	MSB 13	0x28	
	LSB 14	0x2F	
Source Endpoint	15	0xE8	
Destination Endpoint	16	0xE8	
Cluster ID	MSB 17	0x00	
	LSB 18	0x19	
Profile ID	MSB 19	0xC1	
	LSB 20	0x05	
Broadcast radius	21	0x00	

Frame data fields			Offset	Example	Comments
Transmit options			22	0x00	
Data payload	ZCL frame header	Frame control	23	0x09	
		Transaction sequence number	24	0x01	
	ZCL payload	Command ID	25	0x07	Upgrade End response
		Manufacturer ID	LSB 26	0x1E	
			MSB 27	0x10	
		Image type	LSB 28	0x00	
			MSB 29	0x00	
		Firmware version	LSB 30	0x01	
			31	0x10	
			32	0x00	
			MSB 33	0x00	
		Current Time	LSB 34	0xF0	32 bit unsigned integer Seconds since Epoch
			35	0x1A	
			36	0x53	
			MSB 37	0x21	
		Upgrade Time	LSB38	0x00	
39	0x1B				
40	0x53				
MSB 41	0x21				
Checksum			38	0xE5	

OTA error handling

ZCL OTA status code	Value	Description
SUCCESS	0x00	Successful operation
ABORT	0x95	Failed when client or server decides to abort the upgrade process
NOT_AUTHORIZED	0x7E	Server is not authorized to upgrade the client
INVALID_IMAGE	0x96	Invalid OTA upgrade image. For example, the image failed signature validation or CRC.
WAIT_FOR_DATA	0x97	Server does not have data block available yet
NO_IMAGE_AVAILABLE	0x98	No OTA upgrade image available for a particular client
MALFORMED_COMMAND	0x80	The command received is badly formatted or has incorrect parameters
UNSUP_CLUSTER_COMMAND	0x81	Such command is not supported on the device
REQUIRE_MORE_IMAGE	0x99	The client still requires more OTA upgrade image files in order to successfully upgrade

Default response commands

The OTA framework has a command ID **0xB** reserved for error messages that are sent by the target device. Default response commands are transmitted by the target device by wrapping the ZCL payload in a [Explicit Addressing Command frame - 0x11](#). The table below shows the ZCL Payload contents.

Note This is an example for a default response that has been received by an OTA source device. You can see that it is an [Explicit Rx Indicator frame - 0x91](#).

Start Delimiter	8	7E
Length	16	00 17
Frame Type	8	91
Source Address	64	FF FF FF FF FF FF FF FF
Source Address	16	FF FF
Source Endpoint	8	E8
Destination Endpoint	8	E8
Cluster ID	16	00 19
Profile ID	16	C1 05

Receive Options	8	C1
RF Data (ZCL payload. Hex In Little Endian)	Frame Control	00
	Sequence Number	00
	Command ID	0B
	Erring Command	02
	Status	8A
Checksum	F2	

The example above reports an error on the **Query Next Image Response(Erring Command: 0x02)** command informing the server that there is an attempt to update to the same firmware version as the one that is running on the target radio (Status : **0x8A**).

The following table explains the different error statuses which occur at different stages in the OTA upgrade process.

Command ID	ZCL OTA command	Status	XCTU message
0x0B Default Response	0x02 Query Next Image Response	0x80	Incorrect Query Next Image Response Format
		0x85	Attempting to upgrade to invalid firmware (Bad Image Type, Wrong Mfg ID, Wrong HW/SW compatibility(%C))
		0x89	Image size is too big
		0x8A	Please ensure that the image you are attempting to upgrade has a different version than the current version
		0x01	ZCL OTA Message Out of Sequence
0x05 Image Block Response	0x05 Image Block Response	0x80	Incorrect Image Block Response Format
		0x01	ZCL OTA Message Out of Sequence
		0x87	Upgrade File Mismatch
0x08 Upgrade End Response	0x08 Upgrade End Response	0x87	Wrong Upgrade File

When the source device or the server receives a default response frame with a command ID of **0x0B** and the erring command is **0x02** that is, the **Query Next Image Response**, it means there is something wrong with the **Query Next Image Response** sent by the server. Similarly, if the erring command is **0x05** that is, the **Image Block Response**, it means there is something wrong with the **Image Block Response** sent by the server, and the same applies to **Upgrade End Response** where there is an error on the **Upgrade End response** message sent by the server.

Upgrade End Request error statuses

The status field in the [Upgrade End request](#) informs the server of any errors during the download or verification of the OTA firmware update image on the client. The error codes that could be reported

are:

ZCL OTA Command	Status	Error Message
0x06 Upgrade End Request	0x94	Client Timed Out
	0x96	Invalid OTA Image
	0x95	Client Aborted Upgrade
	0x05	Storage Erase Failed
	0x87	Contact Tech Support (Highly unlikely to occur)