Digi XBee3® DigiMesh 2.4

RF Module

User Guide

Revision history—90002277

Revision	Date	Description
A	April 2018	Initial release.

Warranty

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

www.digi.com/howtobuy/terms

Send comments

Documentation feedback: To provide feedback on this document, send your comments to techcomm@digi.com.

Customer support

Digi Technical Support: Digi offers multiple technical support plans and service packages to help our customers get the most out of their Digi product. For information on Technical Support plans and pricing, contact us at +1 952.912.3444 or visit us at www.digi.com/support.

Contents

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

This manual supports the following firmware:

- 3000

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:

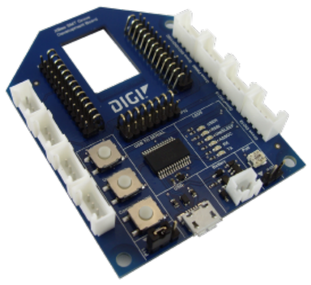
- Zigbee
- 802.15.4
- DigiMesh

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

Getting started

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
- How to unplug an XBee 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

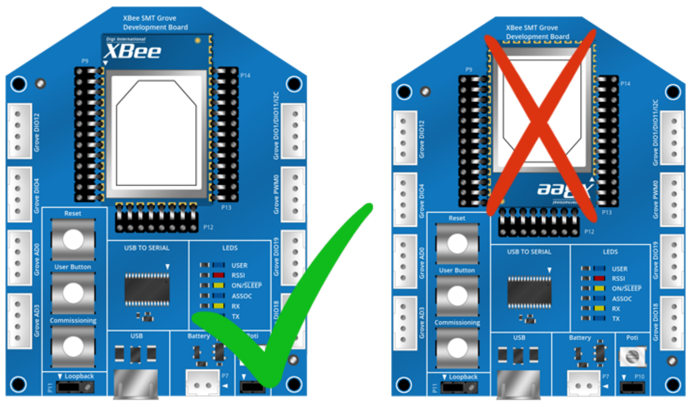
This kit includes three XBee Grove Development Boards. For more information about this hardware, visit the XBee Grove Development Board documentation.

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

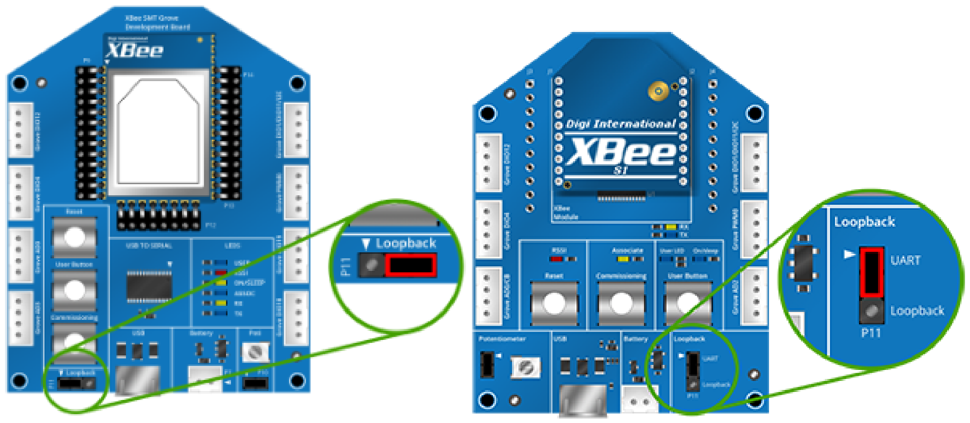
1. Plug one XBee3 DigiMesh RF Module into the XBee Grove Development Board.

   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.



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.



How to unplug an XBee module

To disconnect your XBee module from the XBee Grove Development Board:

1. Disconnect the micro USB cable (or the battery) from the board so it is not powered.
2. Remove the XBee module from the board socket, taking care not to bend any of the pins.

Make sure the board is **not** powered when you remove the XBee 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

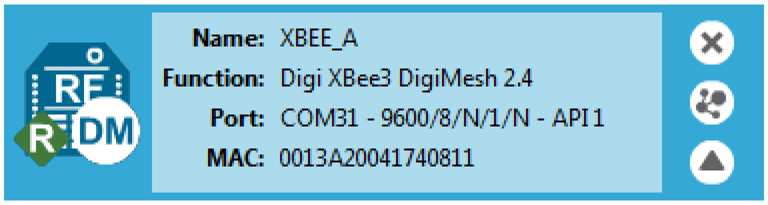
You can communicate with remote devices over the air through a corresponding local device.

Note Configure the local device in API mode because remote commands only work in API mode. Configure remote devices in either API or Transparent mode.

These instructions show you how to configure 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.
3. Configure the first device in API mode and name it **XBEE_A**.
4. Configure the second device in either API or Transparent mode, and name it **XBEE_B**.
5. Disconnect XBEE_B from your computer and remove it from XCTU.
6. Connect XBEE_B to a power supply (or laptop or portable battery).

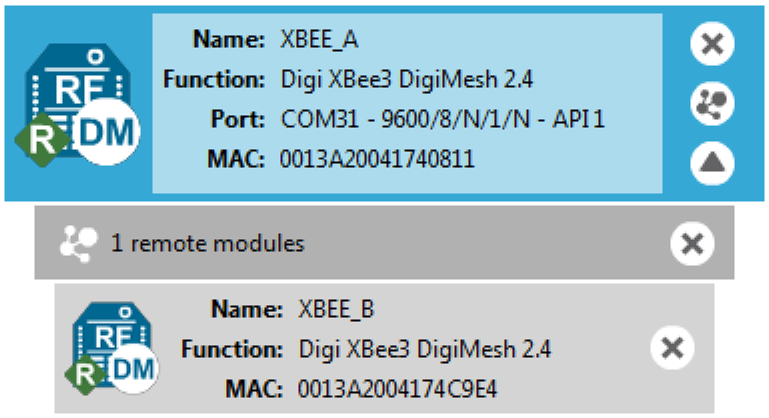
   The **Radio Modules** area should look something like this.



Name: XBEE_A
Function: Digi XBee3 DigiMesh 2.4
Port: COM31 - 9600/8/N/1/N - API 1
MAC: 0013A20041740811

7. Select **XBEE_A** and click the **Discover radio nodes in the same network** button .

8. Click **Add selected devices** in the **Discovering remote devices** dialog. The discovered remote device appears below XBEE_A.



Name:	XBEE_A
Function:	Digi XBee3 DigiMesh 2.4
Port:	COM31 - 9600/8/N/1/N - API 1
MAC:	0013A20041740811

1 remote modules

Name:	XBEE_B
Function:	Digi XBee3 DigiMesh 2.4
MAC:	0013A2004174C9E4

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

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.

1. Add the 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:

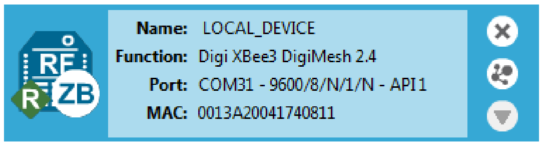
   ID: 2018

   NI: REMOTE_DEVICE
7. Click the **Write radio settings** button.

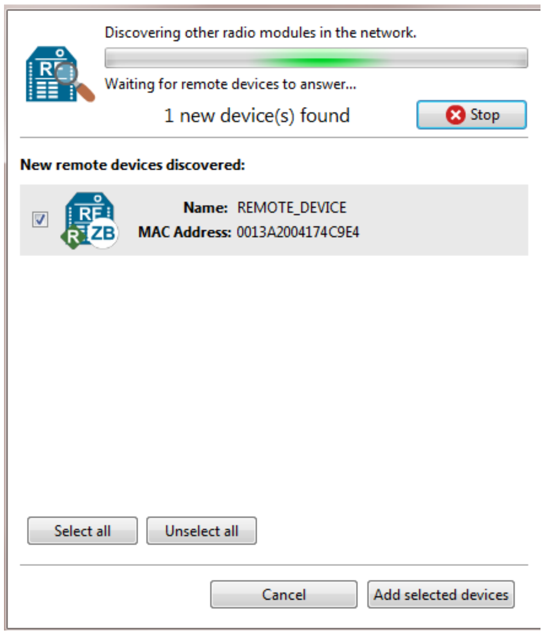
   After you write the radio settings for each device, their names appear in the **Radio Modules** area. The Port indicates that the LOCAL_DEVICE is in API mode.
8. Disconnect REMOTE_DEVICE from the computer, remove it from XCTU, and connect it to its own power supply.
9. Leave LOCAL_DEVICE connected to the computer.

Perform a range test

1. Go to the XCTU display for radio 1.

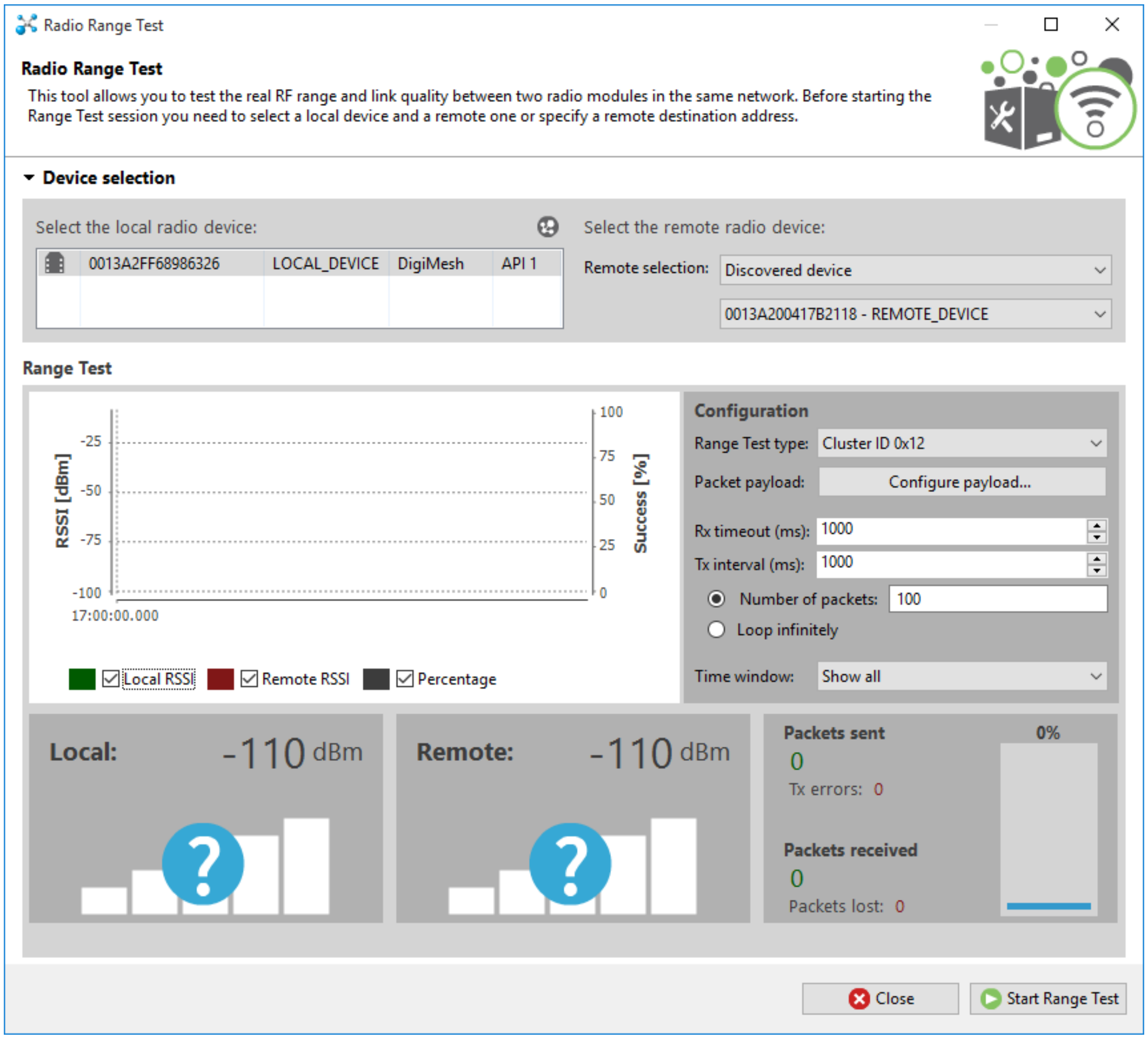

Name: LOCAL_DEVICE
Function: Digi XBee3 DigiMesh 2.4
Port: COM31 - 9600/8/N/1/N - API1
MAC: 0013A20041740811

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


Discovering other radio modules in the network.
Waiting for remote devices to answer...
1 new device(s) found
New remote devices discovered:
Name: REMOTE_DEVICE
MAC Address: 0013A2004174C9E4

3. Click **Add selected devices**.

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



Radio Range Test

Radio Range Test

This tool allows you to test the real RF range and link quality between two radio modules in the same network. Before starting the Range Test session you need to select a local device and a remote one or specify a remote destination address.

Device selection

Select the local radio device:

0013A2FF68986326	LOCAL_DEVICE	DigiMesh	API 1

Select the remote radio device:

Remote selection: Discovered device

0013A200417B2118 - REMOTE_DEVICE

Range Test

Configuration

Range Test type: Cluster ID 0x12

Packet payload: Configure payload...

Rx timeout (ms): 1000

Tx interval (ms): 1000

[✓ Number of packets]: 100

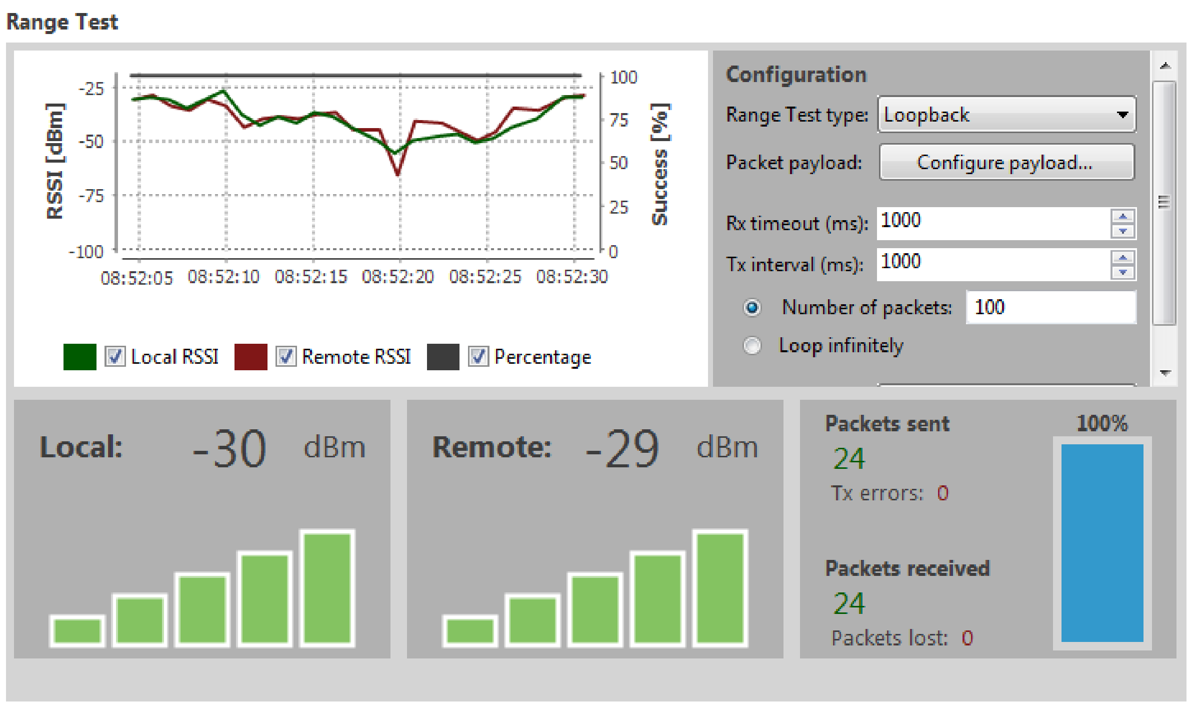
Loop infinitely

Time window: Show all

5. 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.



Configuration
Range Test type: Loopback
Packet payload:
Rx timeout (ms): 1000
Tx interval (ms): 1000
[● Number of packets:] 100
○ Loop infinitely

Local: -30 dBm
Remote: -29 dBm

Packets sent 24
Tx errors: 0
Packets received 24
Packets lost: 0
100%

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. When the test is complete, click **Stop Range Test**.

Configure the XBee3 DigiMesh RF Module

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.

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.

Modes

Transparent operating mode

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

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 device when operating in Transparent mode, you have to enter Command mode and 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.

Command mode is available on the UART interface in both Transparent and API modes. You cannot use the SPI interface to enter Command mode.

Enter Command mode

To get a device to switch into this mode, you must issue the following sequence: **GT** + **CC**(**+++**) + **GT**. When **GT** is set to the default value, if the device sees a full second of silence in the data stream (the guard time) followed by the string **+++** (without Enter or Return) and another full second of silence, it knows to stop sending data through and start accepting commands locally.

Note Do not press Return or Enter after typing **+++** because it will interrupt the guard time silence and prevent 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 Receive 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).

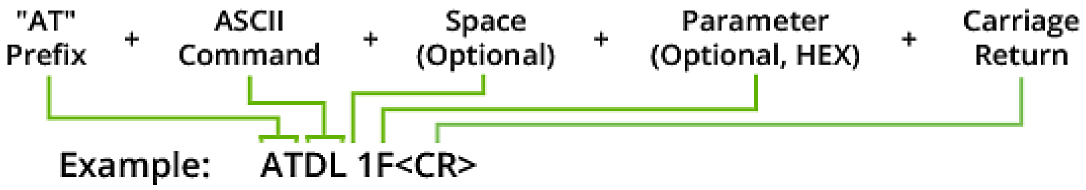
Troubleshooting

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, the **BD** parameter = 3 (9600 baud).

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 the device's destination address (Low) to 0x1F.

To store the new value to non-volatile (long term) memory, send the **WR** (Write) command. This allows parameter values that you modify to persist in the device's registry after a reset. Otherwise, the device restores parameters to the previous values after a reset.

Multiple AT commands

You can send multiple AT commands at a time when they are separated by a comma in Command mode; for example, **ATSH**,**SL**.

Parameter format

Refer to the list of AT commands for the format of individual AT command parameters. Numeric parameters will always be represented in hexadecimal format. Some AT commands have ASCII string parameter, which will be represented as ASCII characters in Command mode and bytes in API mode. Valid formats for hexadecimal values include with or without a leading **0x** for example **FFFF** or **0xFFFF**.

Response to AT commands

When you send a command to the device, the device parses and runs the command. If the command runs successfully, the device returns a response if it is a query. If not a query, the device returns **OK**. If the command errors, the device returns an **ERROR** message.

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 the **AC** (Apply Changes) command.

   or:

2. Exit Command mode.

Exit Command mode

1. Send the **CN** (Exit Command mode) command 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

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

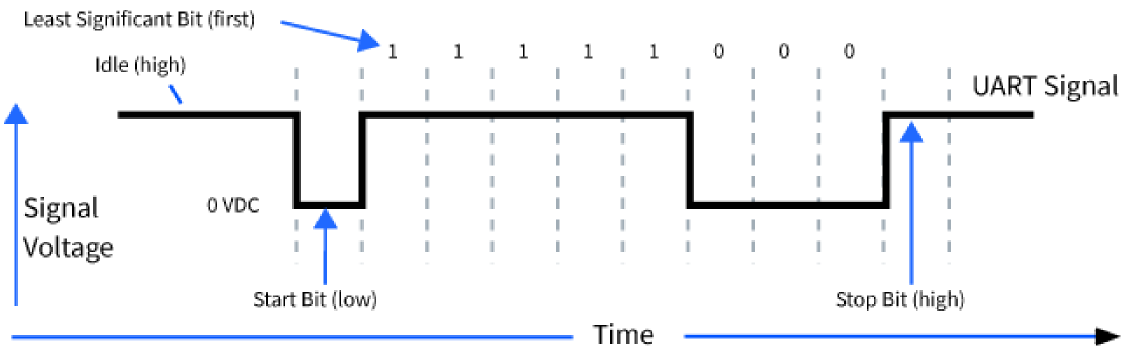
- Through logic and voltage compatible universal asynchronous receiver/transmitter (UART).
- Through a level translator to any serial device, for example, through an RS-232 or USB interface board.

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.

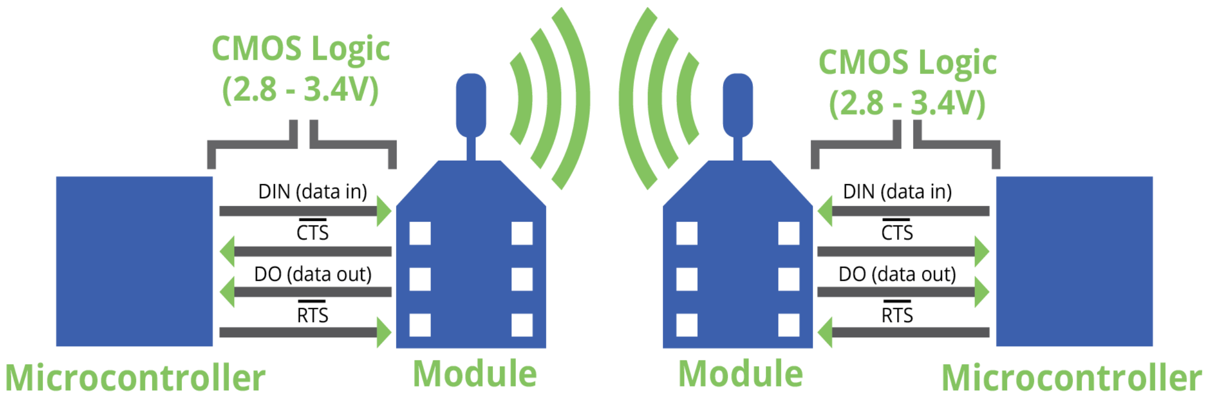
Each data byte consists of a start bit (low), 8 data bits (least significant bit first) and a stop bit (high). The following diagram illustrates the serial bit pattern of data passing through the device. The diagram shows UART data packet 0x1F (decimal number 31) as transmitted through the device.



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 serial interfacing.

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.



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 $\overline{\text{CTS}}$ 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 $\overline{\text{CTS}}$ again, allowing more data to come in.

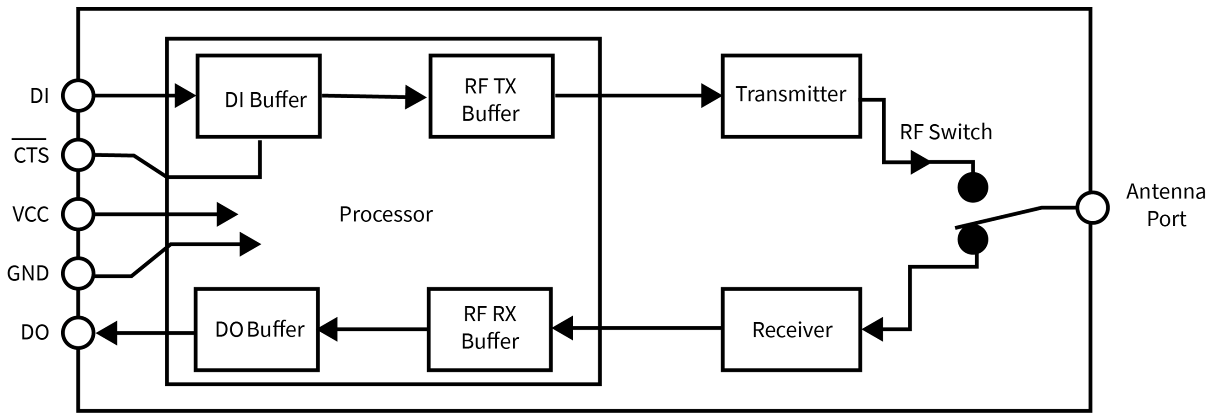
Serial transmit buffer

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.

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) and D7 (DIO7/CTS) to set flow control.



CTS flow control

If you enable CTS flow control (by setting **D7** to 1), when the serial receive buffer is 7 bytes away from being full, the device de-asserts CTS(sets it high) to signal to the host device to stop sending serial data. The device reasserts CTS after the serial receive buffer has 14 bytes of space. The maximum space available for receiving serial data is 109 bytes, which is enough to hold 1.5 full packets of data.

Flow control threshold

Use the **FT** parameter to set the flow control threshold. Since the receive serial buffer is 109 bytes, you cannot set **FT** to more than 109-7 = 102 bytes. This allows up to 7 bytes of data to come in after CTS is de-asserted before data is dropped. The default value of **FT** is 81, leaving space for an external device that responds slowly to CTS being de-asserted. The minimum value of **FT** is 7, which is the minimal operational level.

RTS flow control

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

Networking

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.

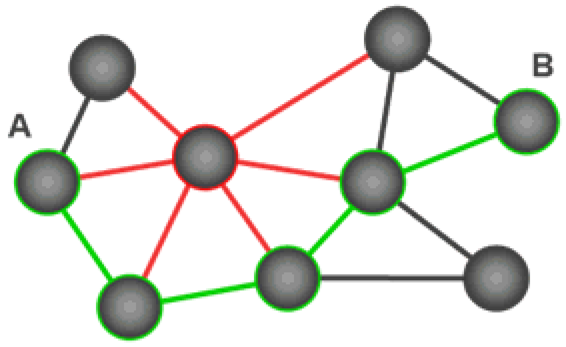
Point-to-multipoint transmissions occur between two adjacent nodes within RF range. No route discovery and no routing occur for these types of transmissions. The networking layer is entirely skipped.

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 determining the plaintext. The XBee3 DigiMesh RF Module performs Electronic Codebook (ECB) mode encryption instead of Counter (CTR) 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

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 0xFFFE. 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.

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

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.

AT commands

Network commands

The following commands are network commands.

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

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)

NT (Network Discovery Back-off)

Sets or displays the network discovery back-off parameter for a device. This sets the maximum value for the random delay that the device uses to send network discovery responses.

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

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.

Parameter range

0 - 2

Parameter	Description	Routes packets
0	Standard router	Yes
2	Non-routing device	No

Default

0

DM (DigiMesh Options)

A bit field mask that you can use to enable or disable DigiMesh 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

Addressing discovery/configuration commands

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

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

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

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

MAC diagnostics

The following AT commands are MAC/PHY commands.

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.

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)

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

N/A

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.

Parameter range

0 - 0xFFFF

Default

N/A

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

Security commands

The following AT commands are security commands.

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

Addressing commands

The following AT commands are addressing commands.

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.

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.

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

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

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

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

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

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

0 - 0xFF

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>
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

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

2

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

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)

Diagnostic commands - addressing timeouts

The following AT commands are diagnostic commands.

%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

RF interfacing commands

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

CH (Operating Channel)

Set or read the operating channel devices used to transmit and receive data. The channel is one of two addressing configurations available to the device. The other configuration is ID (Network ID).

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. Adjacent channel rejection is 23 dB.

The command uses 802.15.4 channel numbers. Center frequency = 2405 MHz + (**CH** - 11 decimal) * 5 MHz.

Parameter range

0xB - 0x1A

Default

0xC (12 decimal)

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

PL setting	XBee3 TX power	XBee3-PRO TX power
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, and negative values are 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, 0x28 - 0x64

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

[read-only]

Default

0

UART serial interfacing

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
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 - 0x03

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

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 - 2

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

Default

0

RO (Packetization Timeout)

Set or read the number of character times of inter-character silence required before transmission begins when operating in Transparent mode.

A “character time” is the amount of time it takes to send a single ASCII character at the operating baud rate (**BD**).

Set **RO** to 0 to transmit characters as they arrive instead of buffering them into one RF packet.

The **RO** command only applies to Transparent mode, it does not apply to API mode.

Parameter range

0 - 0xFF (x character times)

Default

3

FT (Flow Control Threshold)

Set or display the flow control threshold.

The device de-asserts $\overline{\text{CTS}}$ when **FT** bytes are in the UART receive buffer. It re-asserts $\overline{\text{CTS}}$ when less than **FT**-16 bytes are in the UART receive buffer.

Parameter range

0x07 - 0x66 bytes

Default

0x51

Command mode options

The following commands are Command mode option commands.

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)

Diagnostics – Firmware/Hardware Information

The following AT commands are firmware commands.

VR (Firmware Version)

Reads the firmware version on a device.

Parameter range

0x9000 - 0x90FF [read-only]

Default

Set in the firmware

HV (Hardware Version)

Display the hardware version number of the device.

Parameter range

0 - 0xFFFF [read-only]

Default

Set in firmware

DD (Device Type Identifier)

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

If you change **DD**, RE (Restore Defaults) will not restore defaults. The only way to get **DD** back to default values is to explicitly set it to defaults.

Parameter range

0 - 0xFFFFFFFF

Default

0x120000

Note 0x120000 denotes Digi XBee3 hardware.

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

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

Operate in API mode

API mode overview

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

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

Use the AP command to set the operation mode

Use AP (API Enable) to specify the operation mode:

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

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

API frame format

An API frame consists of the following:

- Start delimeter
- Length
- Frame data
- Checksum

API operation (AP parameter = 1)

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

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

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

API operation with escaped characters (AP parameter = 2)

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

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

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

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

Start delimiter field

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

Escaped characters in API frames

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

The following data bytes need to be escaped:

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

To escape a character:

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

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

Example: escape an API frame

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

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

You must escape the 0x13 byte:

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

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

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

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

Length field

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

Frame data

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

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

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

Multi-byte values are sent big-endian.

Calculate and verify checksums

To calculate the checksum of an API frame:

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

To verify the checksum of an API frame:

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

Example

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

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

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

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

Add these hex bytes:

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

Now take the result of 0x247 and keep only the lowest 8 bits which in this example is 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

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 0x08 AT Command frame 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 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

Frame data fields	Offset	Example
AT command	5	0x4E (N)
	6	0x48 (H)
Parameter value (**NH**2 = 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).

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 ID	4	0x01
AT command	5	0x42 (B)
	6	0x44 (D)
Parameter value (**BD**7 = 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

Bit	Meaning	Description
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
Options	16	0x00

Frame data fields	Offset	Example
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.
- 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.
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.

Frame data fields	Offset	Description
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
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

Frame data fields	Offset	Example
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
AT command	5	0x42 (B)
	6	0x44 (D)

Frame data fields	Offset	Example
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 0x0B = Network woke up 0x0C = Network went to sleep

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 a 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 0xFFFD (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

Frame Fields	Offset	Example
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** = **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	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
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
Length	MSB 1	0x00
	LSB 2	0x12

Frame data fields	Offset	Example
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** = 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	Reserved.
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

Node Identification Indicator frame - 0x95

Description

A device receives this frame when:

- it transmits a node identification message to identify itself
- **AO** = **0**

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.	
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	

Frame data fields	Offset	Description
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/AD0).
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
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

Frame data fields	Offset	Example
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
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