XBee® Zigbee® Mesh Kit

Radio Frequency (RF) Module

User Guide

Revision history—90001942-13

Revision	Date	Description
B	January 2016	Updated the documentation to include support for S2C SMT module.
C	June 2016	Updated the document with new Digi branding.
D	August 2017	Added information clarifying that the S2D module is international. Added a graphic of the Worldwide kit and delineated the difference in the Kit Contents table.
E	February 2018	Added information for XBee3 product line and a few minor edits.

Trademarks and copyright

Disclaimers

Warranty

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

www.digi.com/howtobuy/terms

Customer support

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

Product name and model
Product serial number (s)
Firmware version
Operating system/browser (if applicable)
Logs (from time of reported issue)
Trace (if possible)
Description of issue
Steps to reproduce

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

Feedback

To provide feedback on this document, email your comments to

techcomm@digi.com

Include the document title and part number (XBee® Zigbee® Mesh Kit, 90001942-13 D) in the subject line of your email.

Contents

XBee Zigbee Mesh Kit User Guide

Digi’s XBee Zigbee Mesh Kit is a great way to learn how to use XBee RF modules for device connectivity and mesh networking. Starting with very simple examples, we provide step-by-step guidance as you assemble the kit components to create reliable device communications, working control systems, and sensing networks with incredible battery life and robust security.

Mesh networking is a powerful way to route data. Range is extended by allowing data to hop from node to node, and reliability is increased by “self healing,” the ability to create alternate paths when one node fails or a connection is lost. Zigbee is one of the most popular mesh networking protocols, specifically designed for low-data rate and low-power applications. The main advantage of Zigbee is that it is an open standard, so any manufacturer's device that fully supports it can communicate with any other company's Zigbee device.

The kit is designed for anyone getting started in the world of Zigbee. Hardware and software engineers, corporate technologists, or educators and students can quickly create wireless mesh networks.

Each point of this guide explains a basic topic related to XBees through a short theoretical introduction and examples that put into practice the concepts you have learned. The topics are arranged according to their complexity, from the most basic to the more powerful features. We recommend that new users work through them in the order they appear.

This guide provides step by step examples, and some use the Java programming language. These examples are designed to be easy for anyone to use, and those with some programming background can extend them.

Change the firmware protocol

Although the kit comes pre-loaded with Zigbee firmware, you can change the RF protocol used by the XBee 3. To change protocols, use the **Update firmware** feature in XCTU and select the firmware. See the *XCTU User Guide*.

The XBee 3 hardware can run any of the following protocols:

- DigiMesh
- Zigbee
- 802.15.4

For information on each of these firmwares and instructions for how to get started, see the user guide for each protocol:

- DigiMesh
- Zigbee
- 802.15.4

Kit contents

Verify that your kit contains the following components. Then get started by learning about the XBee devices.

Note Some versions of the S2C kit contain three XBee Through-hole technology (THT) Grove Development boards and three XBee THT devices instead of two THT and one Surface-mount technology (SMT).

S2C Zigbee Kit Qty.	XBee3 Zigbee Kit Qty.	Part	
2	-	XBee Grove Development Board	
1	3	XBee Grove Development Board	

S2C Zigbee Kit Qty.	XBee3 Zigbee Kit Qty.	Part	
2	-	XBee Zigbee THT modules (S2C)	
1	-	XBee Zigbee SMT module (S2C)	
-	3	XBee3 Zigbee SMT module	
3	3	Micro USB cables	
-	3	Antenna - 2.4 GHz, half-wave dipole, 2.1 dBi, U.FL female, articulating	

S2C Zigbee Kit Qty.	XBee3 Zigbee Kit Qty.	Part	
2	2	XBee stickers	

XBee Meshkit devices come in two hardware footprints: through-hole and surface mount.

- **Through-hole technology (THT)** XBee devices include the 20-pin socket and require holes for mounting the component on the printed circuit board (PCB), although it is common for the carrier board to contain a female socket.
- **Surface-mount technology (SMT)** XBee devices include 37 pads and are placed directly on the PCB. They do not require holes or sockets for mounting the component.

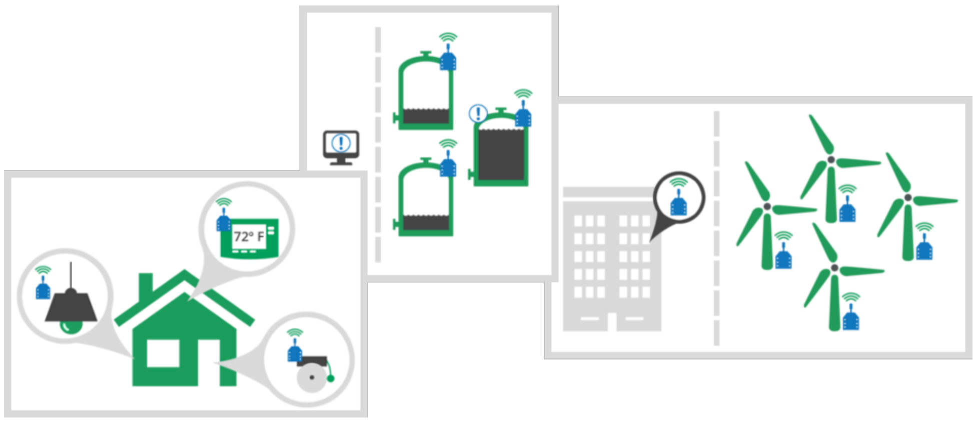
Introduction to XBee devices

XBee modules are small radio frequency (RF) devices that transmit and receive data over the air using radio signals. Wireless capability is essential whenever you want to place sensors where no cables can be installed, or where such tethering is undesirable.

XBee devices are highly configurable and support multiple protocols, which lets you choose the right technology for your application—whether you want to set up a pair of radios to swap data or design a large mesh network with multiple devices.

Here are some of the ways you can use XBee devices:

- Controlling a robot remotely or creating wearable electronics for people, pets, or wildlife, without hindering movement.
- Making a building smarter and more responsive to human interaction.
- Using XBee technology in industrial solutions. For example, XBee devices are used as sensors to monitor industrial tanks for liquid levels, temperature, and pressure, and to monitor and control complex machines such as wind turbines.



Zigbee in a nutshell

Zigbee is an open global standard for low-power, low-cost, low-data-rate, wireless mesh networking based on the IEEE 802.15.4 standard. It represents a network layer above the 802.15.4 layers to support advanced mesh routing capabilities. The Zigbee specification is developed by a growing consortium of companies that make up the Zigbee Alliance. The Alliance is made up of over 300 members, including semiconductor, module, stack, and software developers.

Through its mesh and routing capabilities, Zigbee allows the transmission of data over long distances by passing the data through a mesh network of intermediate nodes to reach more distant nodes. Transmission distance ranges from 1200 to 3200 line-of-sight meters (5280 to 10560 feet). Zigbee supports multiple network topologies such as point-to-point, point-to-multipoint, and mesh networks and allows up to 65,000 nodes per network.

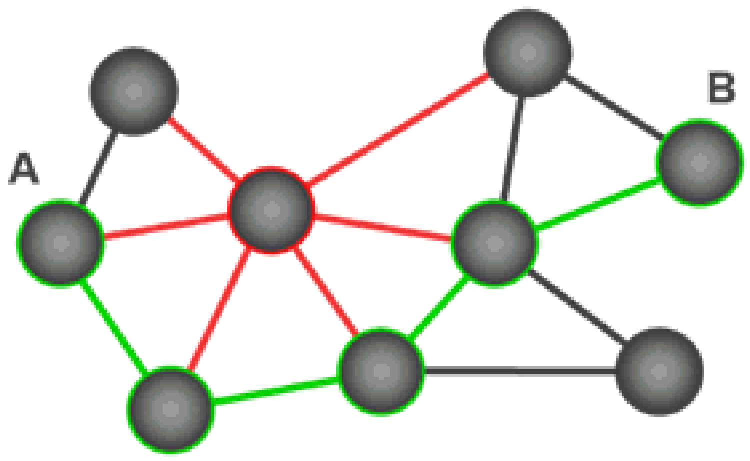
Zigbee is designed to provide the following features:

- High reliability
- Low power consumption
- Low cost
- High security
- Simple protocol, global implementation

Mesh 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 the transmission of information. Mesh networking provides three 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.



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. If a node can no longer operate, for example because it has been removed from the network or because a barrier blocks its ability to communicate, the rest of the nodes can still communicate with each other, either directly or through intermediate nodes.

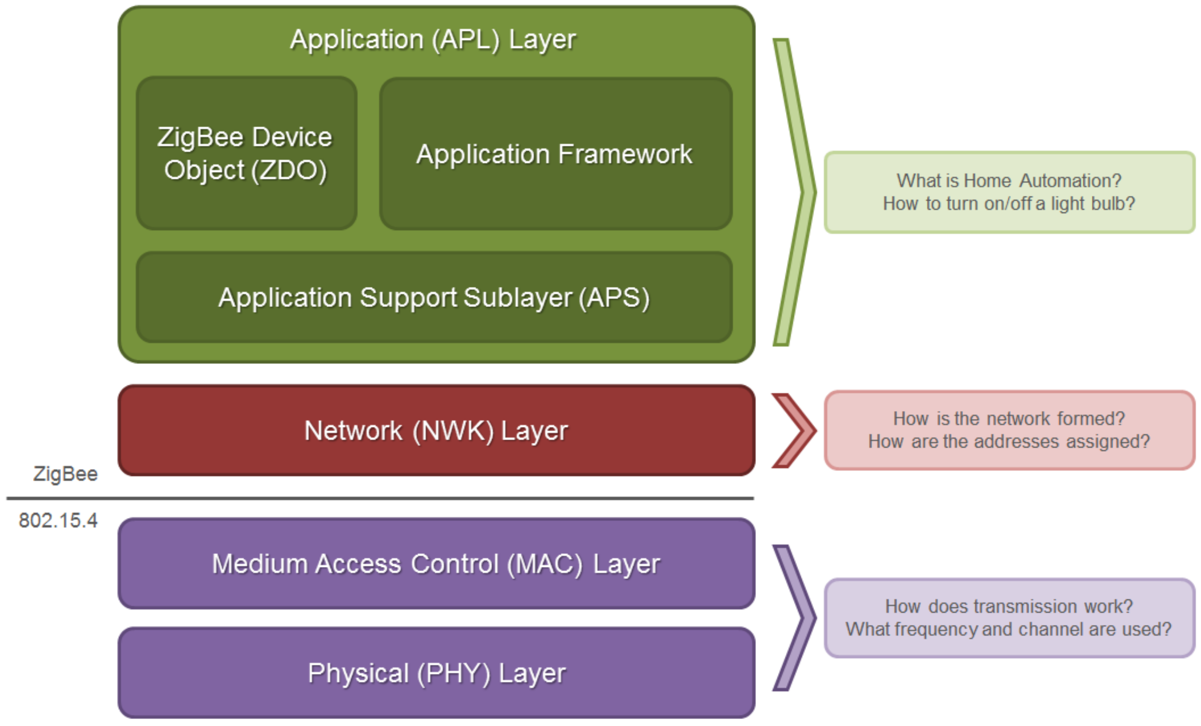
Note Mesh networks use more bandwidth for administration and therefore have less available for payloads. They can also be more complex to configure and debug in some cases.

Zigbee stack layers

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

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

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


Application (APL) Layer
ZigBee Device Object (ZDO)
Application Framework
Application Support Sublayer (APS)
What is Home Automation?
How to turn on/off a light bulb?
Network (NWK) Layer
How is the network formed?
How are the addresses assigned?
ZigBee
802.15.4
Medium Access Control (MAC) Layer
Physical (PHY) Layer
How does transmission work?
What frequency and channel are used?

For more information about the Zigbee stack layers, read the Zigbee communication in depth section.

Device types

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

Coordinator

Zigbee networks always have a single coordinator device. This device:

- Starts the network, selecting the channel and PAN ID.
- Distributes addresses, allowing routers and end devices to join the network. Assists in routing data.
- Buffers wireless data packets for sleeping end device children.
- Manages the other functions that define the network, secure it, and keep it healthy. This device cannot sleep and must be powered on at all times.

Router

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

- Can join existing networks and send, receive, and route information. Routing involves acting as a messenger for communications between other devices that are too far apart to convey information on their own.
- Can buffer wireless data packets for sleeping end device children. Can allow other routers and end devices to join the network.

- Cannot sleep and must be powered on at all times.
- May have multiple router devices in a network.

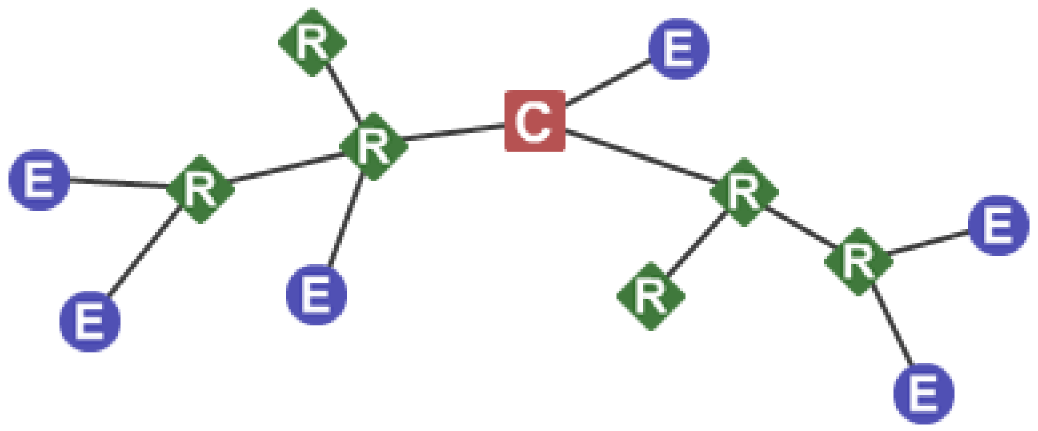
End device

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

- Can join existing networks and send and receive information, but cannot act as messenger between any other devices.
- Cannot allow other devices to join the network.
- Uses less expensive hardware and can power itself down intermittently, saving energy by temporarily entering a non responsive sleep mode.
- Always needs a router or the coordinator to be its parent device. The parent helps end devices join the network, and stores messages for them when they are asleep.

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

An example of such a network is shown in the following diagram:



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

Get started with XBee Zigbee

Use the following steps to set up your environment and assemble the hardware to perform your first XBee application.

Assemble the hardware

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

- Plug in the XBee module
- How to unplug an XBee device

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

Plug in the XBee module

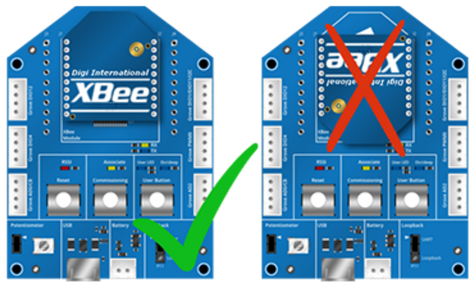
This kit includes several XBee Grove Development Boards. For more information about this hardware, visit the XBee Grove Development Board documentation.

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

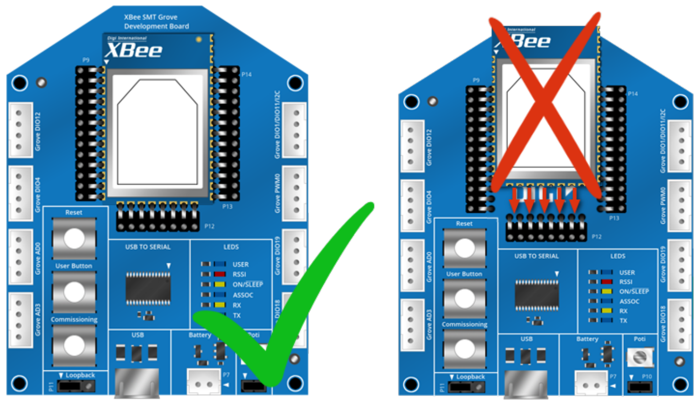
1. Plug one XBee Zigbee Mesh Kit module into the XBee Grove Development Board.

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

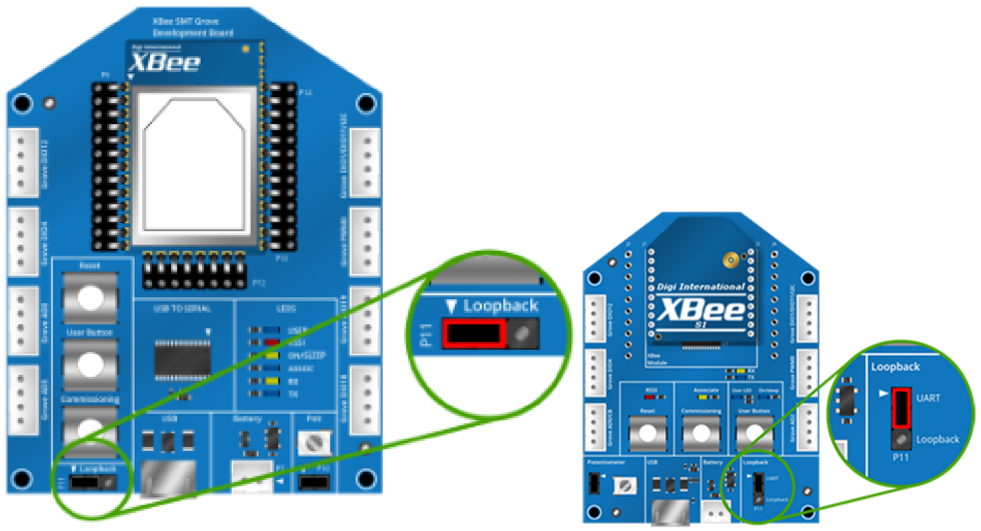
   XBee THT modules have a flat edge and a more angular/diagonal edge. Match that footprint with the white lines on your board and carefully insert it, taking care not to bend any of the pins.



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.



4. Connect an antenna (if applicable).

How to unplug an XBee device

To disconnect your XBee device 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 device from the board socket, taking care not to bend any of the pins.

CAUTION! Make sure the board is **not** powered when you remove the XBee device.

Download and install 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*.

Once you have downloaded XCTU, run the installer and follow the steps to finish the installation process.

After you load XCTU, a message about software updates appears. We recommend you always update XCTU to the latest available version.

Example: basic communication

The goal of this first example is to learn how to set up a simple Zigbee network and transmit data between the nodes. You will assemble the hardware and connect it to your computer, configure the XBees for wireless communication, create a network, and start sending messages.

Use the steps in this section to set up the XBees and send messages using XCTU.

Note Several steps contain videos to help you successfully complete the example. If you get stuck, see Troubleshooting.

Step 1: Requirements

For this setup you need the following hardware and software.

Hardware

- Three XBee Zigbee Mesh Kit modules
- Three XBee Grove Development Boards
- Three micro USB cables
- One computer

Software

- XCTU 6.3.1 or later

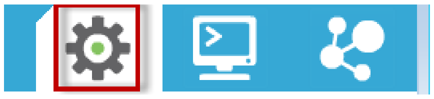
Tip For more information about XCTU, see the XCTU walkthrough.

Step 2: Connect the components

To get started, connect the components and start XCTU.

1. Plug the XBee modules into the XBee Grove Development Boards and connect them to your computer using the micro USB cables provided. For more information, see Plug in the XBee module.
2. After connecting the modules to your computer, open XCTU.

3. Make sure you are in **Configuration working mode**.

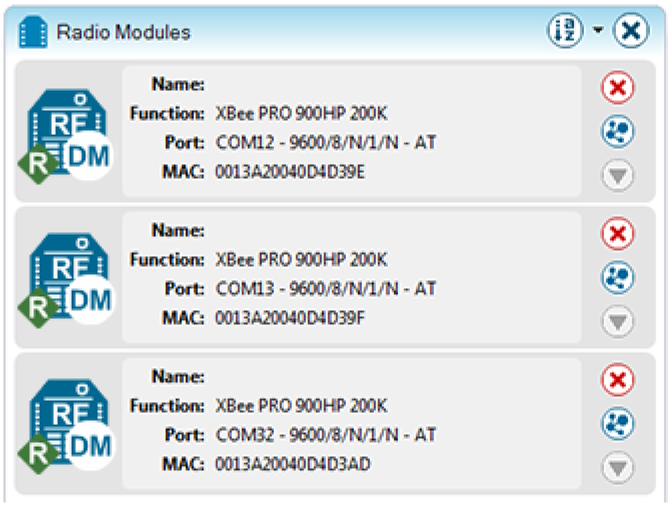


Step 3: Add the XBee modules to XCTU

Use XCTU to find your XBee modules and add them to the tool.

1. Click **Discover radio modules** from the toolbar.

2. In the **Discover radio modules** dialog, select the serial port(s) in which you want to look for radio modules. If you do not know the serial ports where your modules are attached, select all ports. Click **Next**.
3. In the **Set port parameters** window, maintain the default values and click **Finish**.
4. As XCTU locates radio modules, they appear in the **Discovering radio modules**… dialog box. Once the discovery process has finished, click **Add selected devices**.
5. At this point, assuming you have three modules connected to your computer, you should see something like this in the **Radio Modules** section on the left:



Radio Modules

Name:
Function: XBee PRO 900HP 200K
Port: COM12 - 9600/8/N/1/N - AT
MAC: 0013A20040D4D39E

Name:
Function: XBee PRO 900HP 200K
Port: COM13 - 9600/8/N/1/N - AT
MAC: 0013A20040D4D39F

Name:
Function: XBee PRO 900HP 200K
Port: COM32 - 9600/8/N/1/N - AT
MAC: 0013A20040D4D3AD

Note The function, port number and the MAC address displayed for your modules need not match those shown in the picture.

Step 4: Configure the XBee modules

To transmit data wirelessly between your XBee modules, you must configure them to be in the same network. Remember that in Zigbee one device must be the coordinator, and the rest can be routers or end devices. In this case, you will have one router and one end device configured to send data to the coordinator.

1. Restore the default settings of all XBee modules with the **Load default firmware settings** button at the top of the Radio Configuration section.
2. Use XCTU to configure the following parameters:

Param	XBee A	XBee B	XBee C	Effect
ID	2015	2015	2015	Defines the network that a radio will attach to. This must be the same for all radios in your network.
JV	—	Enabled [1]	Enabled [1]	Verifies if a coordinator exists on the same channel to join the network or to leave if it cannot be found.
CE	Enabled [1]	—	—	Sets the device as coordinator.
DH	—	0	0	Defines the destination address (high part) to transmit the data to.
DL	—	0	0	Defines the destination address (low part) to transmit the data to. The address 0000000000000000 can be used to address the coordinator.
NI	COORD	ROUTER	END_ DEVICE	Defines the node identifier, a human-friendly name for the module. ⚠ The default NI value is a blank space. Make sure to delete the space when you change the value.
SP	1F4	1F4	1F4	Defines the duration of time spent sleeping. 1F4 (hexadecimal) = 500 (decimal) x 10 ms = 5 seconds.
SM	—	—	Cyclic sleep [4]	Enables cyclic sleep mode in the end device.
SO	—	—	2	Keeps the module awake during the entire period.

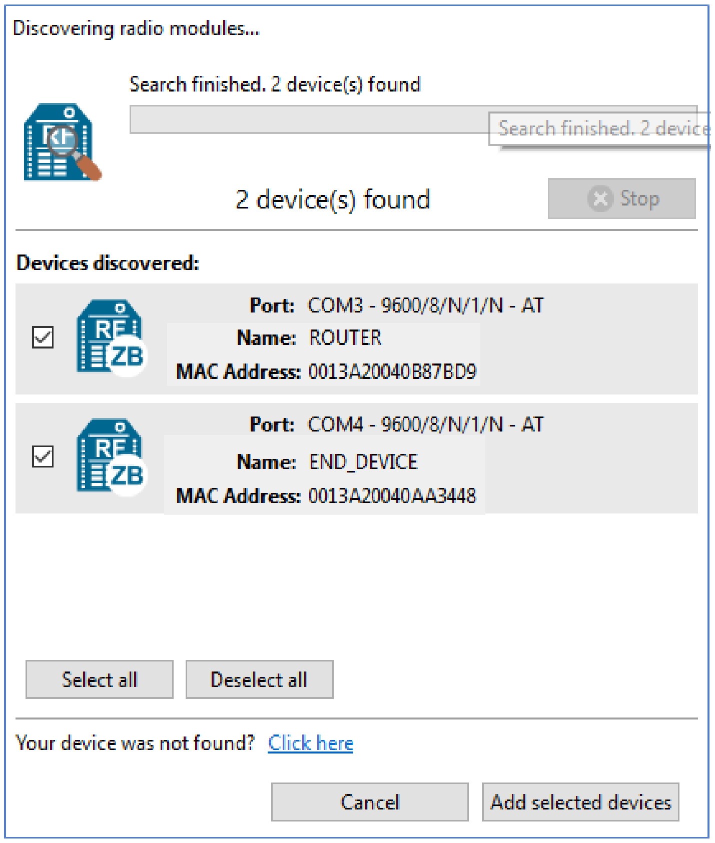
Note The dash (**—**) in the table means to keep the default value. Do not change the default value.

3. Write the settings of all XBee modules with the **Write radio settings** button at the top of the Radio Configuration section.

Step 5: Check the network

Once you have configured your XBee modules, use XCTU to verify that they are in the same network and can see each other.

1. Click the **Discover radio nodes in the same network** button of the first radio module.
   The device searches for radio modules in the same network.



Discovering radio modules...

Search finished. 2 device(s) found

2 device(s) found

Devices discovered:

☑ Port: COM3 - 9600/8/N/1/N - AT
Name: ROUTER
MAC Address: 0013A20040B87BD9

☑ Port: COM4 - 9600/8/N/1/N - AT
Name: END_DEVICE
MAC Address: 0013A20040AA3448

   When the discovery process is finished, XCTU lists discovered devices found within the network in the **Discovering remote devices** dialog. You do not need to add the remote device that has been discovered.
2. Close the dialog by clicking **Cancel**.

Step 6: Send messages

In order to send messages to the coordinator, use the XCTU console or any serial port terminal application such as CoolTerm or TeraTerm (for Windows only). In this case, we will use the XCTU console.

To send messages to the coordinator:

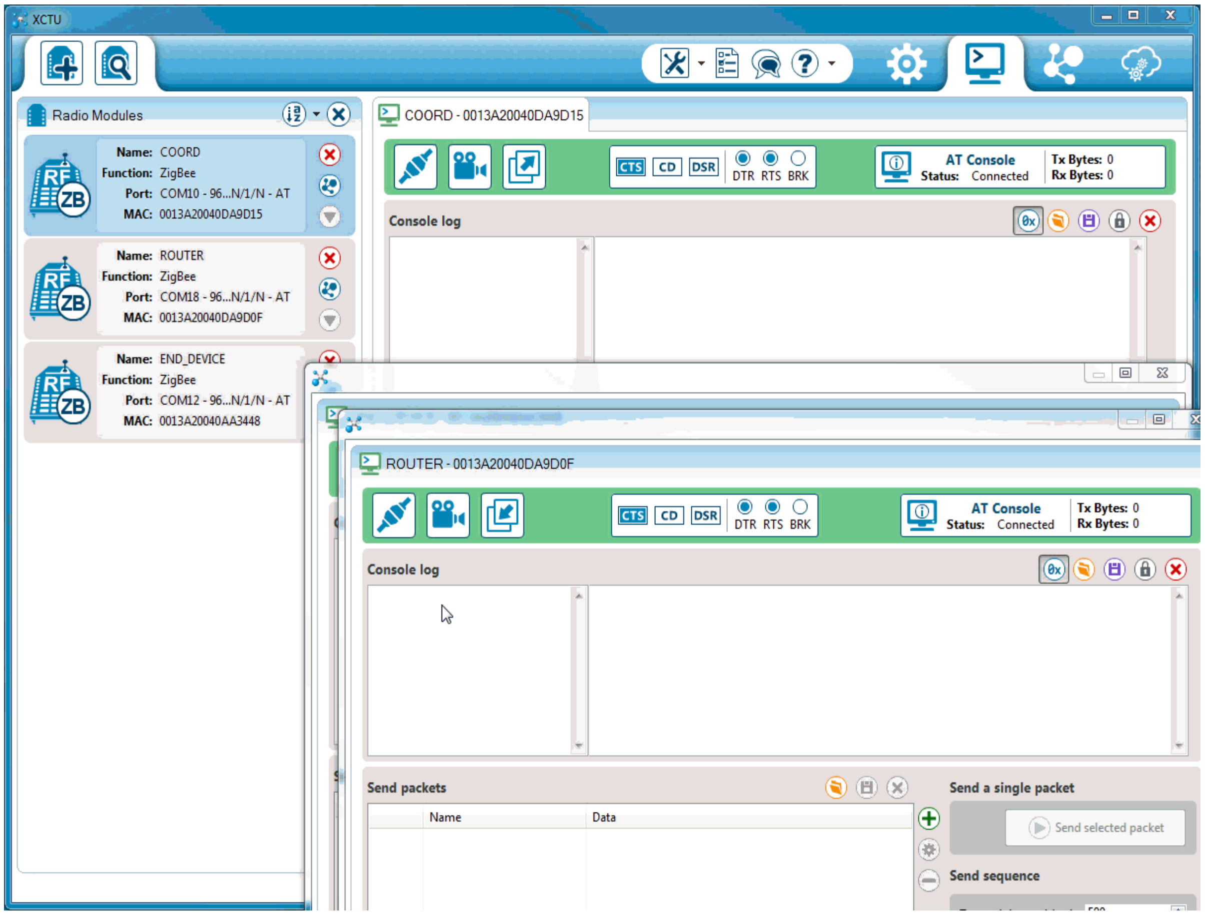
1. If XCTU is not already running, open it.
2. Switch to the Consoles working mode.

   This working mode of XCTU allows you to communicate with the radio modules in the devices list. XCTU loads a list of consoles in the working area—one for each module of the devices list, sorted in a tabbed format.

3. If it is not already there, add an XBee to XCTU so it is listed in the **Radio Modules** list.
4. Then, open the serial connection of the radio module: select the XBee in the **Radio Modules** section, and click the **Open serial connection** button.

   The background changes to green to indicate that the connection is open.
5. Repeat steps 3 and 4 for the others XBee modules.
6. You will see the consoles in three tabs. Click the **Detach view** button to see multiple tabs at the same time.
7. Use the Console log section to type messages.

   Type something like *Hi, this is XXX!* in the ROUTER or END_DEVICE console. The XBee sends every character to COORD and XCTU displays those characters in the corresponding device console.



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

Note If an END_DEVICE is asleep when you type in its console, the message will not be sent to the coordinator. To wake up the module, press the Commissioning button of the XBee Grove Development Board the end device is plugged into.
To identify the END_DEVICE module, look for the board where the On/Sleep LED is ON for five seconds and OFF for another five seconds.

How XBee devices work

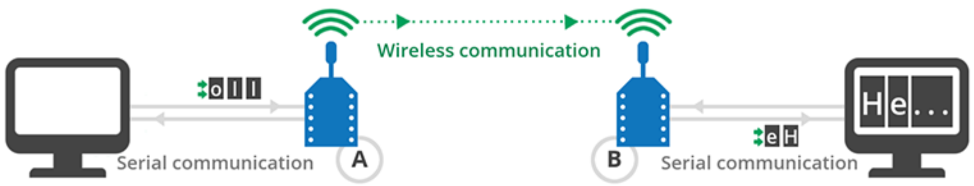
This section describes how XBee devices communicate, and introduces two communication methods - wireless and serial communication. Both communication types are important in the function of XBee devices.

How XBee devices communicate

XBee devices communicate with each other over the air, sending and receiving wireless messages. The devices only transfer those wireless messages; they cannot manage the received or sent data. However, they can communicate with intelligent devices via the serial interface.

XBee devices transmit data coming from the serial input over the air, and they send anything received wirelessly to the serial output. Whether for communication purposes or simply for configuring the device, a combination of both processes makes XBee communication possible. In this way, intelligent devices such as microcontrollers or PCs can control what the XBee device sends and manage incoming wireless messages.

With this information, you can identify the two types of wireless data transmission in an XBee communication process:



1. Wireless communication: This communication takes place between XBee modules. Modules that are supposed to work together need to be part of the same network and they must use the same radio frequency. All modules that meet these requirements can communicate wirelessly with each other.
2. Serial communication: This communication takes place between the XBee module and the intelligent device connected to it through the serial interface.

Wireless communication

XBee modules communicate with each other over the air, transmitting and receiving information via modulation of waves in the electromagnetic spectrum. In other words, they act as radio frequency (RF) devices. For data to transmit from one XBee module to another, both modules must be in the same network.

This section describes the key concepts to understand as you learn how to manage a network and transmit information between XBee modules.

Addressing

XBee device addresses are similar to postal and email addresses for people. Some addresses are unique, like an email address, but others are not. For example, several people can live at the same postal address.

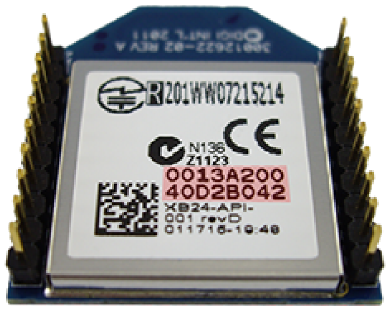
Each XBee device is known by several different addresses, each of which serves a purpose.

Type	Example	Unique
64-bit	0013A20012345678	Always
16-bit	1234	Yes, but only within a network
Node identifier	Bob's module	Uniqueness not guaranteed

64-bit address

Every XBee device has a 64-bit address to distinguish it from others and prevent duplicate information. That address (also called MAC) is assigned to Digi by the IEEE and is guaranteed to be **unique**, so two devices cannot have the same address.

You can determine the value of the 64-bit address by reading the Serial Number High (**SH**) and Serial Number Low (**SL**) parameters on any device. It is also printed on the back of the device.



Note The concatenation of **SH** + **SL** forms the 64-bit or MAC address of the device. It is stored in the device's memory as two 32-bit values: the high part, **SH**, and the low part, **SL**. The high part is usually the same for all XBee devices (0013A200), as this is the prefix that identifies Digi devices. The low part is different for every device.

The 64-bit address of **000000000000FFFF** is reserved for sending a broadcast message.

16-bit address

A device receives a random 16-bit address when it joins a Zigbee network, so this address is also knows as "network address." This address can only change if an address conflict is detected or if a device leaves the network and later joins (it can receive a different address).

The value of the 16-bit address can be read through the 16-bit Network Address (**MY**) parameter. The 16-bit address of 0000 is reserved to the coordinator, while a value of FFFE means the device has not joined a PAN.

Node identifier

The node identifier is a short string of text that allows users to address the module with a more human-friendly name. In this case, uniqueness is not guaranteed because you can assign the same node identifier to several modules.

You can read or set the value of the node identifier through the Node Identifier (**NI**) parameter.

PAN Addresses

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

The value of the personal area network can be set through the PAN ID (**ID**) parameter. If this value is 0, the XBee automatically selects the PAN ID, so you can read it using the Operating PAN ID (**OP**) parameter.

Channels

For the devices to be able to communicate, they must operate in the same frequency. XBee S2C/S2D and XBee3 devices support all 16 channels defined in the 802.15.4 physical layer, with the following exceptions:

- Channel 26 has reduced maximum output power on the S2C/S2D parts (~3dBm).
- S2C XBee-PRO device supports 15 of the 16 channels; it does not support channel 26.
- XBee3-PRO parts support channel 26, but at a reduced maximum output power (~8dBm).

To determine the specific channel where the device is operating, you must read the Operating Channel (**CH**) parameter. Unlike 802.15.4, the **CH** parameter cannot be written in the Zigbee application. However, you can select the operating channel by setting a single bit in the **SC** parameter. That single bit forces a coordinator to operate on the channel specified by the single bit. It also prevents routers and end devices from joining a network on any channel but the one specified in **SC**. If the selected channel is not important, you can use the **SC** parameter to select multiple channels.

Serial communication

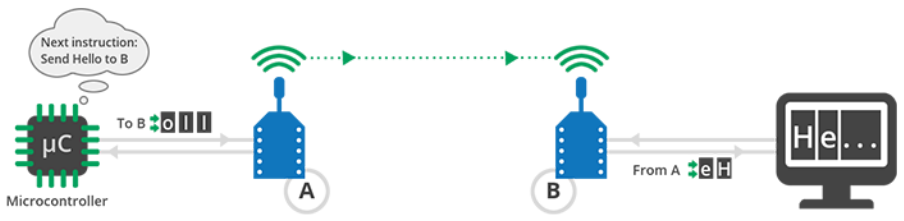
An XBee module can operate as a stand-alone device or it can be attached to an intelligent device. For example, you can place several battery-powered XBee modules in remote locations to gather data such as temperature, humidity, light, or liquid level.

- When operating as a stand-alone device, an XBee module simply sends sensor data to a central node.
- When an XBee module is connected to an intelligent device (such as a computer, Arduino, or Raspberry Pi), it uses serial communication:
  - The intelligent device sends data through the serial interface to the XBee module to be transmitted to other devices over the air.
  - The XBee module receives wireless data from other devices, and then sends the data through the serial interface to the intelligent device.

The XBee modules interface to a host device such as a microcontroller or computer through a logic-level asynchronous serial port. They use a UART for serial communication with those devices.

For additional information about serial communication, go to the XBee/XBee-PRO Zigbee RF Module.

Microcontrollers attached to an XBee module can process the information received by the module and thus monitor or even control remote devices by sending messages through their local XBee module. For prototyping, you can use external microcontrollers such as Arduino or Raspberry Pi, sockets, and breadboards.



The boards included in this kit allow you to use the XBee modules in either mode:

- If you plug the modules into the boards and connect them to a computer or microcontroller using the micro USB cables, you can configure the XBee modules, test the connection, and send/receive data to/from other modules.
- If you plug the modules into the boards and connect them to a battery, the XBee modules work autonomously. For example, they can gather data from a sensor and send it to a central node.

Operating modes

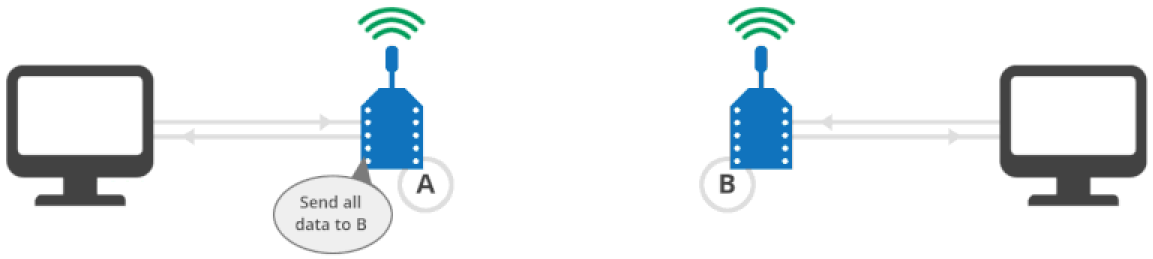
XBee devices can use their local serial connection in very different ways. The "operating mode" establishes the way the host device communicates with an XBee module through the serial interface.

XBee modules support two different operating modes:

- Application Transparent ("transparent mode")
- Application Programming Interface ("API mode")

Application Transparent operating mode

This mode is called "transparent" because the radio passes information along exactly as it receives it. All serial data received by the radio module is sent wirelessly to a remote destination XBee module. When the other module receives the data, it is sent out through the serial port exactly as it was received. Transparent mode has limited functionality but is an easy way to get started with XBee devices.



To learn more about transparent mode, see XBee transparent mode.

API operating mode

Application Programming Interface (API) operating mode is an alternative to transparent mode. In API mode, a protocol determines the way information is exchanged. Data is communicated in packets (commonly called API frames). This mode allows you to form larger networks and is more appropriate for creating sensor networks to perform tasks such as collecting data from multiple locations, controlling devices remotely, or automating your home.

To learn more about API mode, see API mode.

Comparison of transparent and API modes

XBee devices can use transparent or API operating mode to transmit data over the serial interface. You can use a mixture of devices running API mode and transparent mode in a network. The following table provides a comparison of the two modes.

Transparent operating mode	API operating mode
When to use: ■ Conditions for using API mode do not apply.	**When to use:** ■ Sends wireless data to multiple destinations. ■ Configures remote XBee devices in the network. ■ Receives wireless data packets from multiple XBee devices, and the application needs to identify which devices send each packet. ■ Receives I/O samples from remote XBee devices. ■ Must support multiple endpoints, clusters, and/or profiles (for Zigbee modules). ■ Uses Zigbee Device Object (ZDO) services (for Zigbee modules).
Advantages: ■ Provides a simple interface that makes it easy to get started with XBee devices. ■ Easy for an application to support; what you send is exactly what other modules get, and vice versa. ■ Works very well for two-way communication between XBee devices.	**Advantages:** ■ Can set or read the configuration of remote XBee devices in the network. ■ Can transmit data to one or multiple destinations; this is much faster than transparent mode where the configuration must be updated to establish a new destination. ■ Received data includes the sender's address. ■ Received data includes transmission details and reasons for success or failure. ■ Several advanced features, such as advanced networking diagnostics, and firmware upgrades.

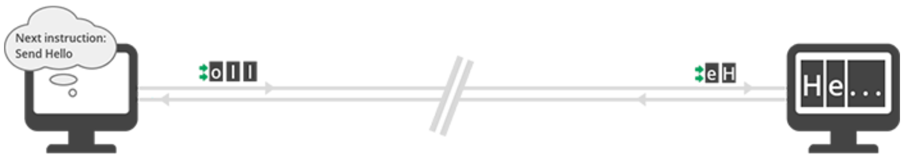
Transparent operating mode	API operating mode
Disadvantages: ■ Cannot set or read the configuration of remote XBee devices in the network. ■ Must first update the configuration to establish a new destination and transmit data. ■ Cannot identify the source of received data, as it does not include the sender's address. ■ Received data does not include transmission details or the reasons for success or failure. ■ Does not offer the advanced features of API mode, including advanced networking diagnostics, and firmware upgrades.	**Disadvantages:** ■ Interface is more complex; data is structured in packets with a specific format. ■ More difficult to support; transmissions are structured in packets that need to be parsed (to get data) or created (to transmit data). ■ Sent data and received data are not identical; received packets include some control data and extra information.

XBee transparent mode

This section provides additional detail about XBee transparent mode. For a comparison of transparent and API modes, see Serial communication.

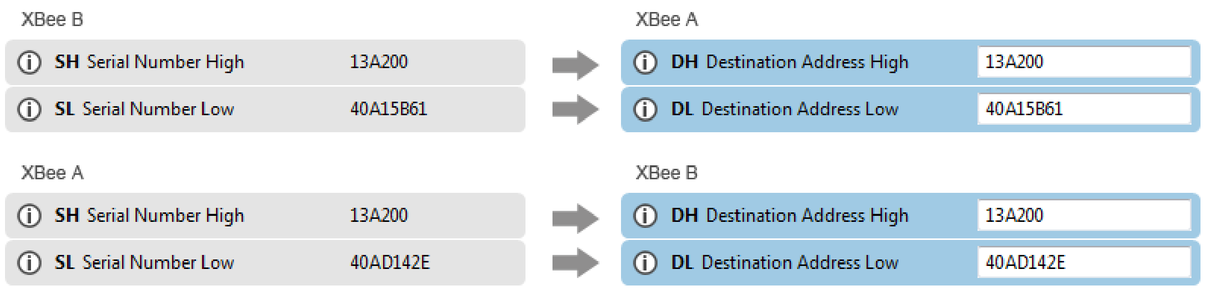
XBee transparent mode in detail

When operating in transparent mode, an XBee module acts as a serial line replacement. All data received through the serial input is immediately transmitted over the air. When the XBee module receives wireless data, it is sent out through the serial interface exactly at it is received. In fact, communication in transparent mode yields the same result as if the two modules were connected by a wire, but wireless communication makes that physical wire unnecessary.



For two XBee modules to communicate, the sending module needs the address of the recipient. When working in transparent mode, you must configure this address in the module that is communicating. XBee modules can store the complete 64-bit address of the destination module. This address must be programmed in two parameters: Destination Address High (**DH**) and Destination Address Low (**DL**).

If you want modules A and B to communicate, configure the destination address (**DH** + **DL**) of XBee A as the MAC address (**SH** + **SL**) of XBee B, and vice versa.



XBee B

SH	Serial Number High	13A200
SL	Serial Number Low	40A15B61

XBee A

DH	Destination Address High	13A200
DL	Destination Address Low	40A15B61

XBee A

SH	Serial Number High	13A200
SL	Serial Number Low	40AD142E

XBee B

DH	Destination Address High	13A200
DL	Destination Address Low	40AD142E

Transparent mode has some limitations. For example, when you are working with several modules, you must configure the destination before sending each message. However, transparent mode provides an easy way to get started with XBee devices for the following reasons:

- Operation is very simple.
- What you send is exactly what the other modules get.
- Compatible with any device that can communicate over a serial interface.
- Works very well when facilitating communication between two XBee modules.

What have you learned?

- An XBee communicates remotely with other XBees via wireless and locally with the intelligent device (microcontroller, computer) connected to it via the serial interface.
- To communicate wirelessly, your modules must be part of the same network, so the **ID** and **CH** parameters must have identical values for all XBees in the network.
- Every XBee module has a unique 64-bit address called MAC that distinguishes it from the rest of the devices. This address is formed by the concatenation of the parameters **SH** (Serial Number High) and **SL** (Serial Number Low).
- The operating mode of an XBee establishes the way to communicate with the module through the serial interface.

- There are two different operating modes: Transparent and API (Application Programming Interface).
- Transparent mode can be used as a serial cable replacement. What is sent through an XBee serial input is wirelessly received by the destination module and then sent out to its serial output exactly as it was transmitted from the first XBee (and vice versa).
- In order to communicate using transparent mode, you must pre-configure each of your devices by setting the parameters **DH** and **DL** on the first module with the **SH** and **SL** values of the other one respectively, and vice versa.

Extend the basic communication example

If you're ready to move beyond this exercise and extend the example, try the following:

- Use Arduino or Raspberry Pi instead of a computer to transmit data wirelessly.
- Use your XBees as a cable replacement for your serially communicating applications. For example, if you have an Arduino application that controls room lighting via the serial port, replace the serial cable with XBees and move your Arduino around the house.
- Try using XBees to send wireless messages to your friends and neighbors.

Command mode

An XBee device in Transparent mode simply passes information along exactly as it receives it. So, what you send is what other devices get. But sometimes you want to talk directly to the local device without sending data. For example, you may need to modify its configuration or alter the way it behaves. In that case, the XBee device needs to know that this communication should not be transmitted wirelessly.

Command mode is a state in which incoming characters are interpreted as commands. To get a device to switch into this mode, you must issue a unique string of text in a special way: **+++**. When the device sees a full second of silence in the data stream 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.

Guard time silence	Command sequence	Guard time silence
One second before	+++	One second after

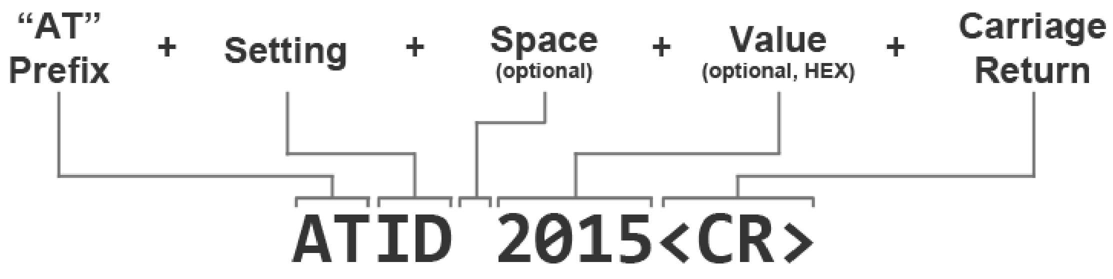
Do not press Return or Enter after typing the **+++** because it will interrupt the guard time silence and prevent the module from entering Command mode.

Once the device is in Command mode, it listens for user input for a while. If 10 seconds go by without any user input, the device automatically drops out of Command mode and returns to Transparent mode.

AT commands

The purpose of command mode is to read or change the configuration of the local XBee device. Every module has a number of settings, like channel or network ID, that define its behavior. These settings are identified by two characters, for example, **CH** for channel, and **ID** for network ID.

When you want to read or set any setting of the XBee **module**, you must send it an AT command. Every AT command starts with the letters "AT" followed by the two characters that identify the command being issued and then by some optional configuration values.



For example, to read and set the network ID setting:

```
// Enter command mode
+++OK

// Read the ID setting
ATID <Enter>
0

// Change the ID setting
ATID 2015 <Enter>
OK
```

Basic AT commands

- **AT**

  This command checks the connection with the module. This is like asking "Are you there?" and the device replying "Yes." When you send this command, the module simply replies OK. If you don't see an OK in response, you have probably timed out of command mode. Type the **+++** to go back into it.

- **ATCN**

  This command explicitly exits the module from command mode. Remember that if you don't type anything for 10 seconds, the device automatically drops out of Command mode.

- **ATWR**

  This command writes the current configuration to non-volatile memory so that it persists the next time the device powers up. Otherwise, parameters are restored to previously saved values after the device is reset.

Use AT commands

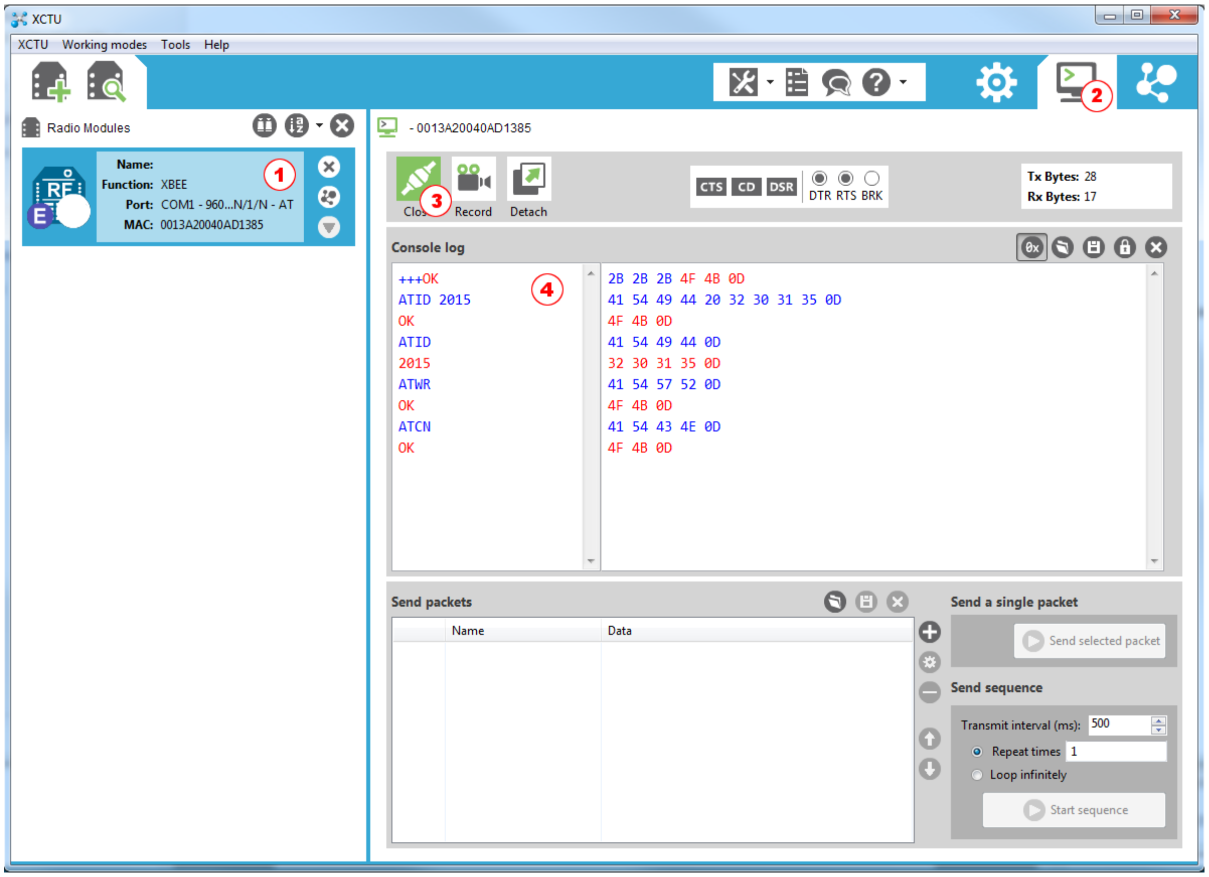
In the first example in this kit, you used XCTU to configure some settings of each of your modules, such as the network ID. XCTU uses AT commands in the background to read and set the settings. For example, when you changed the value of that parameter and clicked the Write button, XCTU went into command mode using **+++,** changed the value of the setting with the **ATID** command, wrote the setting with the **ATWR**command, and finally exited command mode with the `ATCN` command.

XCTU simplifies the configuration of the XBee modules so you don't have to use command mode or AT commands to configure them. However, you can always configure an XBee module through any serial port terminal application or the XCTU console.

The following example demonstrates how you can perform some of the configuration steps outlined in the first lab but via command mode and using AT commands:

1. In the Consoles working mode of XCTU, click the **Open the serial connection with the radio module** button.
2. Use **+++** to enter into command mode and wait for an OK response.
3. To set a register, type an AT command followed by the value you want to set; for example, **ATID 2015**; followed by a Return.
4. To read a register, type an AT command; for example, **ATID**; followed by a Return.
5. Use the **ATWR** command to write the new configuration to the module's memory.
6. Exit command mode with the **ATCN** command.

Note You should get an **OK** response after issuing each command to set parameters, write the changes, or exit from command mode. If not, you most likely took more than 10 seconds to issue the command and you have dropped out of command mode.



Name:
Function: XBEE
Port: COM1 - 960...N/1/N - AT
MAC: 0013A20040AD1385

Tx Bytes: 28
Rx Bytes: 17

```
+++OK       2B 2B 2B 4F 4B 0D
ATID 2015   41 54 49 44 20 32 30 31 35 0D
OK          4F 4B 0D
ATID        41 54 49 44 0D
2015        32 30 31 35 0D
ATWR        41 54 57 52 0D
OK          4F 4B 0D
ATCN        41 54 43 4E 0D
OK          4F 4B 0D
```

Transmit interval (ms): 500
Repeat times 1

API mode

This section provides additional detail about API mode and lets you put your knowledge into practice. For a comparison of transparent and API modes, see Serial communication.

API mode in detail

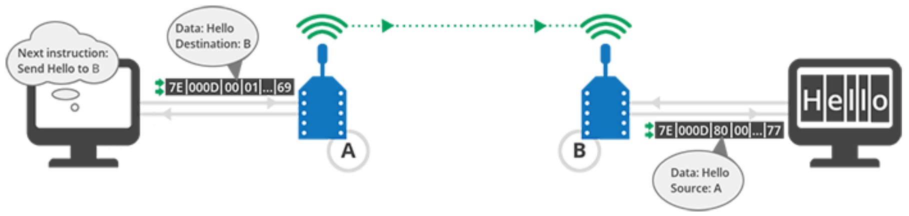
API mode provides a structured interface where data is communicated through the serial interface in organized packets and in a determined order. This enables you to establish complex communication between devices without having to define your own protocol.

By default, XBee devices are configured to work in transparent mode: all data received through the serial input is queued up for radio transmission and data received wirelessly is sent to the serial output exactly as it is received, with no additional information.

Because of this behavior, devices working in Transparent mode have some limitations:

1. To read or write the configuration of an device in Transparent mode, you must first transition the device into Command mode.
2. If a device needs to transmit messages to different devices, you must update its configuration to establish a new destination. The device must enter Command mode to set up the destination.
3. A device operating in Transparent mode cannot identify the source of a wireless message it receives. If it needs to distinguish between data coming from different devices, the sending devices must include extra information known by all the devices so it can be extracted later. To do this, you must define a robust protocol that includes all the information you think you need in your transmissions.

To minimize the limitations of the transparent mode, devices provide an alternative mode called Application Programming Interface (API). 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 modules without having to define your own protocol.



API mode provides a much easier way to perform the actions listed above:

1. Since there are different frames for different purposes (such as configuration and communication), you can configure a device without entering Command mode.
2. Since the data destination is included as part of the API frame structure, you can use API mode to transmit messages to multiple devices.
3. The API frame includes the source of the message so it is easy to identify where data is coming from.

Advantages of API mode

- Configure local and remote XBee devices in the network.
- Manage wireless data transmission to one or multiple destinations.
- Identify the source address of each received packet.
- Receive success/failure status of each transmitted packet.

- Obtain the signal strength of any received packet.
- Perform advanced network management and diagnosis.
- Perform advanced functions such as remote firmware update, ZDO, ZCL and so on.

API frame structure

The structured data packets in API mode are called frames. They are sent and received through the serial interface of the device and contain the wireless message itself as well as some extra information such as the destination/source of the data or the signal quality.

When a device is in API mode, all data entering and leaving the module through the serial interface is contained in frames that define operations or events within the device.

An API frame has the following structure:

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

Note MSB represents the most significant byte, and LSB represents the least significant byte.

Any data received through the serial interface prior to the start delimiter is silently discarded by the XBee. If the frame is not received correctly, or if the checksum fails, the data is also discarded and the module indicates the nature of the failure by replying with another frame.

Start delimiter

The start delimiter is the first byte of a frame consisting of a special sequence of bits that indicate the beginning of a data frame. Its value is always 0x7E. This allows for easy detection of a new incoming frame.

Length

The length field specifies the total number of bytes included in the frame data field. Its two-byte value excludes the start delimiter, the length, and the checksum.

Frame data

This field contains the information received or to be transmitted. Frame data is structured based 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	Frame-type-specific data							Single byte

Note MSB represents the most significant byte, and LSB represents the least significant byte.

- **Frame type** is the API frame type identifier. It determines the type of API frame and indicates how the information is organized in the Data field.
- **Data** contains the data itself. The information included here and its order depends on the type of frame defined in the Frame type field.

Checksum

Checksum is the last byte of the frame and helps test data integrity. It is calculated by taking the hash sum of all the API frame bytes that came before it, excluding the first three bytes (start delimiter and length).

Note Frames sent through the serial interface with incorrect checksums will never be processed by the module and the data will be ignored.

Calculate the checksum of an API frame

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

Example: Checksum calculation

To calculate the checksum for the given frame:

Start Delimiter	Length		Frame Data															Checksum
			Frame type	Data														
7E	00	0F	17	01	00	13	A2	00	40	AD	14	2E	FF	FE	02	44	42	-

1. Add all bytes excluding the start delimiter and the length: 17 + 01 + 00 + 13 + A2 + 00 + 40 + AD + 14 + 2E + FF + FE+ 02 + 44 + 42 = 481
2. From the result, keep only the lowest 8 bits: 81.
3. Subtract that result from 0xFF: FF - 81 = 7E

In this example, 0x7E is the checksum of the frame.

Verify the checksum of a given 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 will equal FF.

Example: Checksum verification

In our example above, we want to verify the checksum is 7E.

			Frame Data															
Start Delimiter	Length		Frame type	Data														Checksum
7E	00	0F	17	01	00	13	A2	00	40	AD	14	2E	FF	FE	02	44	42	7E

1. Add all data bytes and the checksum: 17 + 01 + 00 + 13 + A2 + 00 + 40 + AD + 14 + 2E + FF + FE + 02 + 44 + 42 + 7E = 4FF
2. Since the last two far right digits of 4FF are FF, the checksum is correct.

Supported frames

Support for API frame types depends on the type of XBee you are using. The 802.15.4 XBee modules included in this kit support the following API frames:

Transmit data frames are sent through the serial input, with data to be transmitted wirelessly to remote XBees:

API ID	Frame name	Description
0x08	AT Command	Queries or sets parameters on the local XBee
0x09	AT Command Queue Parameter Value	Queries or sets parameters on the local XBee without applying changes
0x10	Transmit Request	Transmits wireless data to the specified destination
0x11	Explicit Addressing Command Frame	Allows Zigbee application layer fields (endpoint and cluster ID) to be specified for a wireless data transmission
0x17	Remote AT Command Request	Queries or sets parameters on the specified remote XBee module
0x21	Create Source Route	Creates a source route in the module
0x24	Register Joining Device	Registers a module with the Trust Center

Receive data frames are received through the serial output, with data received wirelessly from remote XBees:

API ID	Frame name	Description
0x88	AT Command Response	Displays the response to previous AT command frame
0x8A	Modem Status	Displays event notifications such as reset, association, disassociation, and so on.
0x8B	Transmit Status	Indicates wireless data transmission success or failure
0x90	Receive Packet	Sends wirelessly received data out the serial interface (AO = 0)
0x91	Explicit Rx Indicator	Sends wirelessly received data out the serial interface when explicit mode is enabled (AO 0)
0x92	IO Data Sample Rx Indicator	Sends wirelessly received IO data out the serial interface
0x94	XBee Sensor Read Indicator	Sends wirelessly received sensor sample (from a Digi 1-wire sensor adapter) out the serial interface
0x95	Node Identification Indicator	Displays received node identification message when explicit mode is disabled (AO = 0)

API ID	Frame name	Description
0x97	Remote AT Command Response	Displays the response to previous remote AT command requests
0x98	Extended Modem Status	Displays what is happening during the association when Verbose Join is enabled (DC10)
0xA0	Over-the-Air Firmware Update Status	Provides a status indication of a firmware update transmission attempt
0xA1	Router Record Indicator	Displays the multiple route hopes after a Zigbee route record command
0xA3	Many-to-One Route Request Indicator	Indicates a many-to-one route request is received
0xA5	Join Notification Status	Indicates a module attempts to join, rejoin, or leave the network

For more information about the structure of some of these frames, see the XBee/XBee-PRO Zigbee RF Module User Guide.

Frame examples

The following examples of sent and received API frames are expressed in hexadecimal format.

Example: 0x10 - Transmit Request

The following frame is a Transmit Request frame with the following characteristics:

7E 00 13 10 01 00 13 A2 00 40 DA 9D 23 A6 B9 00 00 48 65 6C 6C 6F 0C

- The frame ID is 0x01, so the sender will receive a Transmit Status frame with the result of the transmission.
- The destination XBee has a 64-bit address of **00 13 A2 00 40 DA 9D 23** and 16-bit address of **A6 B9**.
- It does not specify any option.
- The data to transmit is 'Hello' (**48 65 6C 6C 6F**).

Frame fields		Offset	Example	Description
Start delimeter		0	0x7E	
Length		MSB 1	0x00	Number of bytes between the length and the checksum
		LSB 2	0x13	

Frame fields		Offset	Example	Description
Frame data	Frame type	3	0x10	0x10 - Indicates this is a *Transmit Request* frame
	Frame ID	4	0x01	Identifies the data frame for the host to correlate with a subsequent *Transmit Status (0x8B)* frame. Setting Frame ID to '0' will disable response frame.
	64-bit Destination address	MSB 5	0x00	Set to the 64-bit address of the destination XBee The following addresses are also supported: ■ 0x0000000000000000 - Coordinator address ■ 0x000000000000FFFF - Broadcast address ■ 0xFFFFFFFFFFFFFFFF - Unknown address if the destination's 64-bit address is unknown
		6	0x13	
		7	0xA2	
		8	0x00	
		9	0x40	
		10	0xDA	
		11	0x9D	
		LSB 12	0x23	
	16-bit Destination address	MSB 13	0xA6	Set to the 16-bit address of the destination XBee, if known. The following addresses are also supported: ■ 0x0000 - Coordinator address ■ 0xFFFE - Unknown address if the destination's 16-bit address is unknown, or if sending a broadcast
		LSB 14	0xB9	
	Broadcast Radius	15	0x00	Sets the maximum number of hops a broadcast transmission can occur. If set to '0', the broadcast radius will be set to the maximum hops value.

Frame fields		Offset	Example	Description
	Options	16	0x00	Bitfield of supported transmission options Supported values include the following: ■ 0x01 - Disable retries ■ 0x20 - Enable APS encryption (if EE = 1) ■ 0x40 - Use the extended transmission timeout for this destination All other bits must be set to 0. Enabling APS encryption decreases the maximum number of RF payload bytes by 4 (below the value reported by NP). Setting the extended timeout bit causes the stack to set the extended transmission timeout for the destination address.
	RF Data	MSB 14	0x48	Up to 255 bytes of data that is sent to the destination XBee
		15	0x65	
		...	0x6C	
		17	0x6C	
		LSB 18	0x6F	
Checksum		22	0x6E	Hash sum of frame data bytes

Example: 0x91 - Explicit Rx Indicator

The following frame is an Explicit Rx Indicator frame with the following characteristics:

7E 00 17 91 00 13 A2 00 40 DA 9D 05 00 00 E8 E8 00 11 C1 05 01 48 65 6C 6C 6F 61

- The XBee module that sent this data has a 64-bit address of **00 13 A2 00 40 DA 9D 05** and 16-bit address of **00 00**.
- The endpoint of the source that initiated the transmission is **E8** and the destination endpoint is **E8**.
- The Cluster ID the data is addressed to is **00 11**.
- The Profile ID the data is addressed to is **C1 05**.

- The packet was acknowledged because the Receive options value is **01**.
- The received data 'Hello' is (**48 65 6C 6C 6F**).

Frame fields		Offset	Example	Description
Start delimiter		0	0x7E	
Length		MSB 1	0x00	Number of bytes between the length and the checksum
		LSB 2	0x17	

Frame fields		Offset	Example	Description
Frame data	Frame type	3	0x91	0x91 - Indicates this is a *Explicit Rx Indicator* frame
	64-bit Source Address	MSB 4	0x00	64-bit address of sender Set to 0xFFFFFFFFFFFFFFFF (unknown 64-bit address) if the sender's 64-bit address is unknown
		5	0x13	
		6	0xA2	
		7	0x00	
		8	0x40	
		9	0xDA	
		10	0x9D	
		LSB 11	0x05	
	16-bit Source Network Address	MSB 12	0x00	16-bit address of sender
		LSB 13	0x00	
	Source Endpoint	14	0xE8	Endpoint of the source that initiated the transmission
	Destination Endpoint	15	0xE8	Endpoint of the destination the message is addressed to
	Cluster ID	16	0x00	Cluster ID the message was addressed to
		17	0x11	
	Profile ID	18	0xC1	Profile ID the message was addressed to
		19	0x05	

Frame fields		Offset	Example	Description
	Receive Options	20	0x01	Bitfield of supported transmission options Supported values include the following: ▪ 0x01 - Packet Acknowledged ▪ 0x02 - Packet was a broadcast packet ▪ 0x20 - Packet encrypted with APS encryption ▪ 0x40 - Packet sent with extended timeout enabled
	Received Data	MSB 21	0x48	Up to 255 bytes data received from the source XBee
		22	0x65	
		…	0x6C	
		24	0x6C	
		LSB 25	0x6F	
Checksum		26	0x61	Hash sum of frame data bytes

Operating mode configuration

The API Enable (**AP**) parameter configures the XBee module to operate using a frame-based API instead of the default Transparent mode. It allows you to select between the two supported API modes and the default transparent operation.

Mode	AP value	Description
Transparent	0	API modes are disabled and the module operates in transparent mode
API 1	1	API mode without escaped characters
API 2	2	API mode with escaped characters

The only difference between API 1 and API 2 is that API 2 operating mode requires that frames use escape characters (bytes).

Configuration of the serial XBee communication—whether it is transparent, API non-escaped (API 1), or API escaped (API 2)—does not prevent wireless communication between XBee modules. Since only the payload portion of the API frame is transmitted over the air, the receiving XBee modules will alter the packet information based on their **AP** setting, allowing an API non-escaped module to successfully communicate with others working in API escaped or Transparent mode.

Note Devices working in Transparent mode and modules set to API non-escaped (API 1) operation can communicate with devices configured to work in API escaped mode (API 2).

API escaped operating mode (API 2)

API non-escaped (API 1) operation relies solely on the start delimiter and length bytes to differentiate API frames. If bytes in a packet are lost, the length count will be off, and the next API frame (packet) will also be lost. API escaped (API 2) operation involves escaping character sequences in an API frame in order to improve reliability, especially in noisy RF environments.

The basic frame structure of both API modes is the same, but in API escaped (API 2) mode, all bytes except for the start delimiter must be escaped if needed. The following data bytes must be escaped in API 2 mode:

- 0x7E: Start delimiter
- 0x7D: Escape character
- 0x11: XON
- 0x13: XOFF

API 2 mode guarantees all the 0x7E bytes received are start delimiters: this character cannot be part of any of the other frame fields (length, data, or checksum) since it must be escaped.

To escape a character:

1. Insert 0x7D, the escape character.
2. Append it with the byte to be escaped, XORed with 0x20.

In API 2 mode, the length field does not include any escape character in the frame and the checksum is calculated with non-escaped data.

Example: Escape an API frame

To express the following API non-escaped frame in API 2 mode:

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

The 0x13 byte must be escaped:

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

This is the resulting frame. Note that the length and checksum are the same as the non-escaped frame.

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

XBee frame exchange

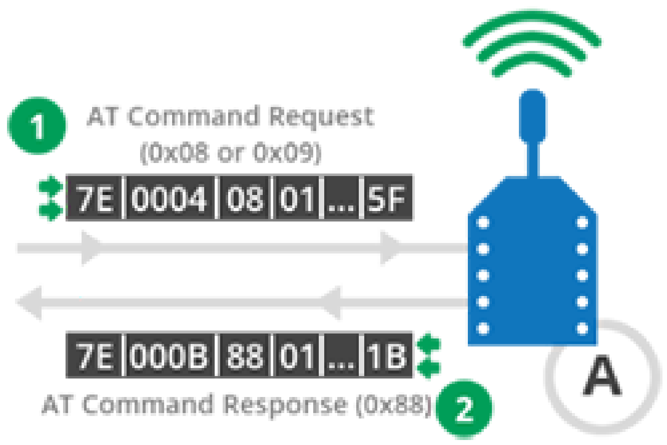
Now that you understand how API mode works and how API frames are structured, the next step is to learn about how frames are exchanged when you perform certain common operations such as configuring an XBee module or transmitting wireless data.

The following section provides examples using XCTU. You can use the Frames interpreter tool of the XCTU console to view detailed API frame structure.

AT Command: configure a local XBee device

To query or set the value of the local XBee—that is, the device directly connected to an intelligent device such as a microcontroller or PC via the serial interface—you must use AT parameters and commands. These are the same AT parameters and commands that are available in Transparent/Command mode, but included in an AT Command (0x08) frame. The response containing the result of the operation is sent back in an AT Command Response (0x88) frame.

The following image shows the API frame exchange that takes place at the serial interface when sending either an AT Command (0x08) or an AT Command Queue Parameter Value (0x09) request.



During an API frame exchange, the following process occurs:

1. An AT Command (0x08) frame is sent to the device through the serial input. This frame contains configuration instructions or queries parameters on the local XBee device.
2. The XBee device processes the command and returns an AT Command Response (0x88) through its serial output. If the frame ID of the AT Command frame is **0**, this response is not sent.

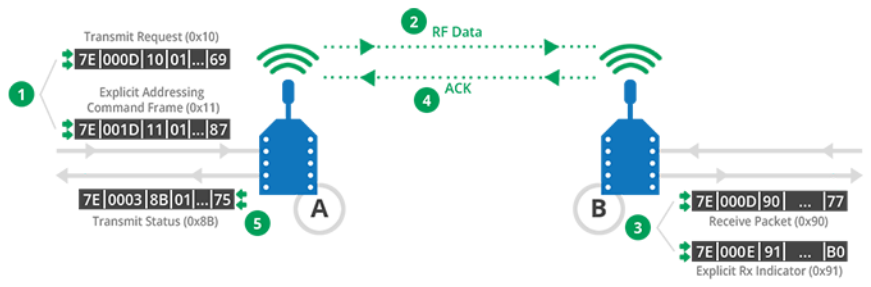
Transmit Request/Receive Packet: Transmit and receive wireless data

A Transmit Request frame encapsulates data with its remote destination and some transmission options. The wireless data received by an XBee module is included in a Receive Packet frame along with the remote transmitter and options for receipt.

Two more frames use Explicit addressing. They require that you specify application layer addressing fields (endpoints, cluster ID, profile ID).

For more information about Explicit addressing, see the Zigbee communication in depth chapter.

The following image shows the API exchanges that take place at the serial interface when transmitting wireless data to another XBee module.



1. The intelligent device (host) sends a Transmit Request (0x10) or an Explicit Addressing Command Frame (0x11) to XBee A through the serial input to transmit data to XBee B.
2. XBee A wirelessly transmits the data in the frame to the module configured as destination in the same frame; in this case, the destination is XBee B.
3. The remote XBee B module receives the wireless data and sends out through the serial output a Receive Packet (0x90) or an Explicit Rx Indicator (0x91), depending on the value of API Options (**AO**) setting. These frames contain the data received over the air and the source address of the XBee module that transmitted it, in this case XBee A.
4. The remote XBee B module transmits a wireless acknowledge packet with the status to the sender, XBee A.
5. The sender XBee A module sends out a Transmit Status (0x8B) through its serial output with the status of the transmission to XBee B.

The Transmit Status (0x8B) frame is always sent at the end of a wireless data transmission unless the frame ID is set to '0' in the transmit request. If the packet cannot be delivered to the destination, the transmit status frame will indicate the cause of failure.

To send data using an explicit frame:

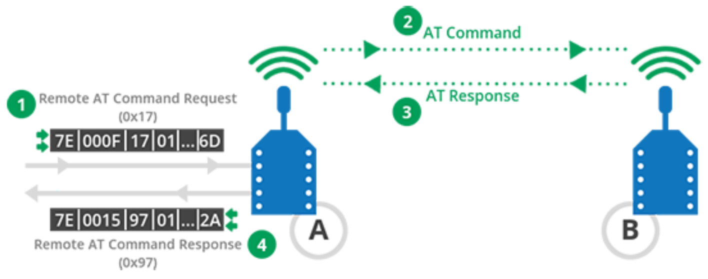
- The source and destination endpoints must be E8.
- The cluster ID must be 0011.
- The profile ID must be C105.

To receive an explicit frame, the API Options (**AO**) parameter must be configured to API Explicit Rx Indicator - 0x91 [1]. If this setting is API Rx Indicator - 0x90 [0], a Receive Packet (0x90) will be received instead of an Explicit Rx Indicator (0x91).

Remote AT Command: Remotely configure an XBee module

Working in API mode also allows you to configure remote XBee modules wirelessly. Any AT command or parameter that can be issued locally can also be sent wirelessly for execution on a remote XBee module.

The following image shows the API frame exchanges that take place at the serial interface when sending a Remote AT Command Request (0x17) to remotely read or set an XBee parameter.


1 Remote AT Command Request (0x17)
7E 000F 17 01 ... 6D
2 AT Command
3 AT Response
7E 0015 97 01 ... 2A
4 Remote AT Command Response (0x97)
A
B

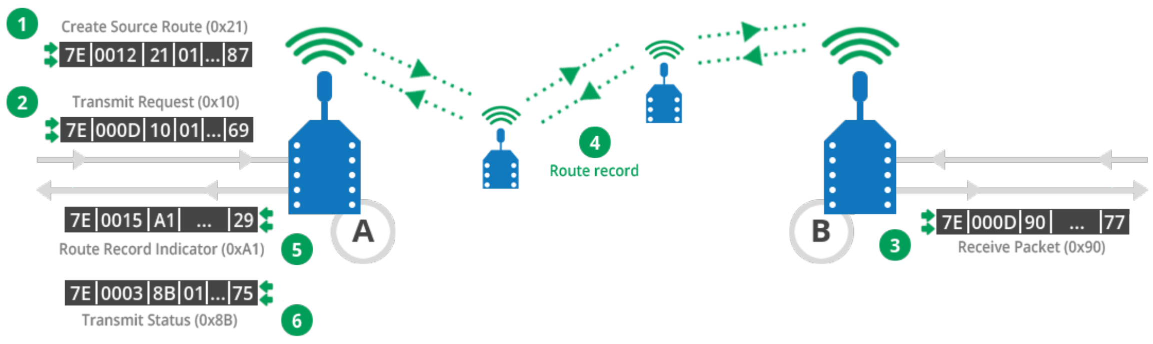
1. The intelligent device (host) sends a Remote AT Command Request (0x17) to XBee A through the serial input to configure the remote XBee B.
2. XBee A wirelessly transmits the AT Command in the frame to the module configured as destination in the same frame; in this case, the destination is XBee B.
3. XBee B receives the AT command and processes the command to wirelessly return the result to the sender, XBee A.
4. XBee A sends out a Remote AT Command Response (0x97) through its serial output with the result of the AT command processed by XBee B. If the frame ID of the Remote AT Command frame is '0', this response is not sent.

Source routing: Create and obtain the route of a packet

XBee modules also allow you to create and obtain a source route in the module. A source route specifies the complete route a packet travels to get from source to destination. In this case, before sending the data (a Transmit Request or a Explicit Addressing Command Frame), you must send a a Create Source Route (0x21) with the route of the following packet.

Use source routing with many-to-one routing for best results, so you should set the Many-to-One Route Broadcast Time (AR) parameter to a value other than FF in the sender module.

The following image shows the API exchanges at the serial interface when you are sending a Create Source Route (0x21) frame.



1. The intelligent device (host) sends a Create Source Route (0x21) to XBee A through the serial input to specify the route of the following data to XBee B.
2. The intelligent device (host) sends a Transmit Request (0x10) or an Explicit Addressing Command Frame (0x11) to XBee A through the serial input to transmit data to XBee B.
3. The remote XBee B receives the wireless data and sends out through the serial output a Receive Packet (0x90) or an Explicit Rx Indicator (0x91). This frame contains the data

received over the air and the source address of the XBee that transmitted it, in this case XBee A.

4. The remote XBee B transmit a route record to the sender, XBee A, with the route the transmitted data followed.
5. The sender XBee A sends out a Route Record Indicator (0xA1) through its serial output with the 16-bit addresses of the nodes the received route record traversed.
6. The sender XBee A sends out a Transmit Status (0x8B) through its serial output with the status of the transmission to XBee B.

Example: Configure your local XBee module

This section demonstrates how to read the Node Identifier (NI) of your local XBee module configured in API mode. To do this, you create an AT command frame to read the NI parameter, send it to the XBee module, and analyze the response.

If you get stuck, see Troubleshooting.

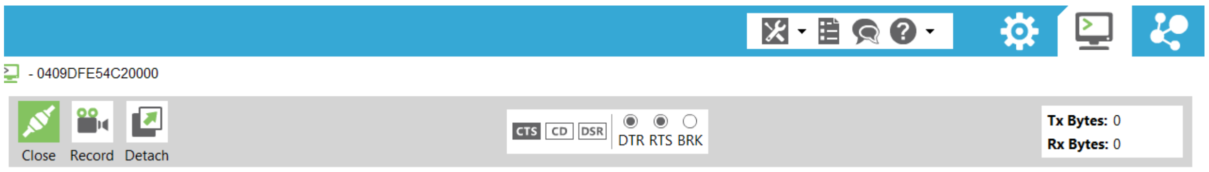
Step 1: Configure the XBee module

Before creating and sending the frame, configure the XBee module as follows:

Param	Value	Effect
NI	XBEE_A	Defines the node identifier, a human-friendly name for the module. ⚠ The default **NI** value is a blank space. Make sure to delete the space when you change the value.
AP	AP Enabled [1]	Enables API mode.

Step 2: Open the XCTU console

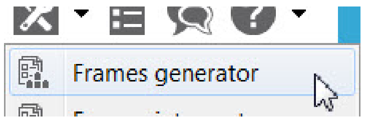
1. Switch to the **Consoles working mode** .
2. Open the serial connection with the radio module .



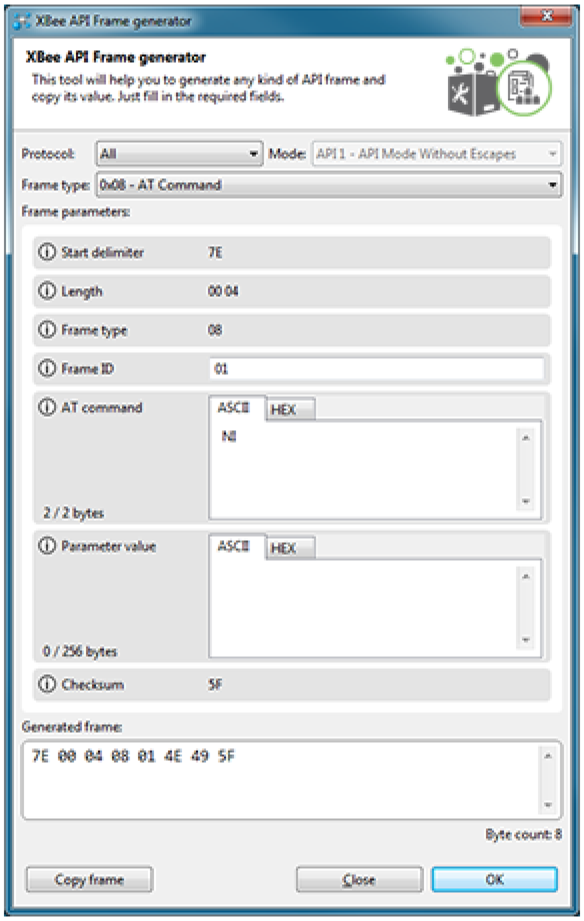
Step 3: Generate the AT command frame

These instructions describe how to generate an AT command frame using the XCTU Frame Generator tool.

1. Click **Add new frame to the list** ⊕.
2. Open the **Frames Generator** tool.



3. In the **Frame type** section, select **0x08 - AT Command**.
4. In the **AT command** section, select the **ASCII** tab and type **NI**.
5. Click **OK**.
6. Click **Add frame**.


XBee API Frame generator

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

Protocol: All
Mode: API 1 - API Mode Without Escapes
Frame type: 0x08 - AT Command
Frame parameters:

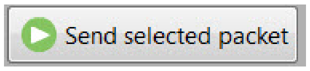
Start delimiter	7E
Length	00 04
Frame type	08
Frame ID	01
AT command (2 / 2 bytes)	ASCII: NI
Parameter value (0 / 256 bytes)	
Checksum	5F

Generated frame:
```
7E 00 04 08 01 4E 49 5F
```
Byte count: 8

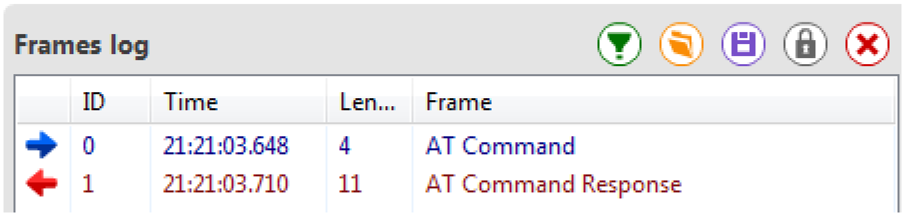
Step 4: Send the AT command frame

After you have created an AT command frame, you must send it to the local XBee module to receive a response containing the configured **NI** value.

1. Select the frame in the XCTU **Send frames** section.
2. Click **Send selected packet**.



The **Frames log** indicates that one frame has been sent (blue) and another has been received (red).



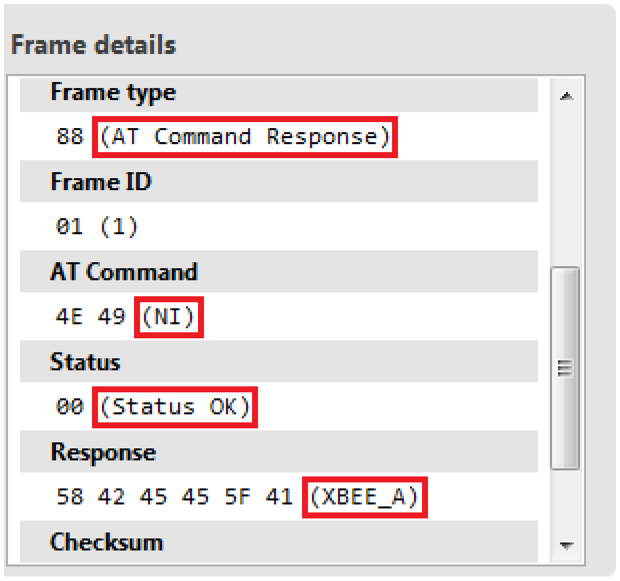
Frames log

	ID	Time	Len...	Frame
→	0	21:21:03.648	4	AT Command
←	1	21:21:03.710	11	AT Command Response

Step 5: Analyze the response

Once you have sent the frame, you can analyze the responses on the receiving end.

1. Select the frame received (AT Command Response) to see its details in the **Frame details** section.
2. Analyze its details and verify that it contains the **NI** value of your module.
   - **Frame type**: The received frame is an AT Command Response.
   - **Frame ID**: This AT Command Response frame is the answer to the sent AT Command request because both have the same value (1).
   - **Status**: The value was successfully read because the status is OK.
   - **Response**: This received frame contains the value of the NI parameter previously requested in the AT Command frame, XBEE_A



Frame details

```
Frame type
88 (AT Command Response)
Frame ID
01 (1)
AT Command
4E 49 (NI)
Status
00 (Status OK)
Response
58 42 45 45 5F 41 (XBEE_A)
Checksum
```

3. Disconnect the console by clicking **Close the serial connection** .

Example: Transmit and receive data

This section describes how to transmit data to another XBee module using the XCTU console. The steps include creating a Transmit Request frame with the message you want to transmit to the other module and sending the frame serially to the local XBee module. You can then analyze the responses, both in the local and the remote module.

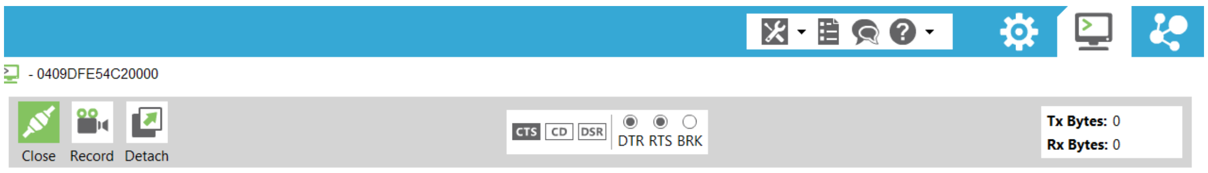
If you get stuck, see Troubleshooting.

Step 1: Configure the XBee modules

Before creating and sending the frame, configure the XBee modules as follows:

Param	XBee A	XBee B	Effect
ID	2015	2015	Defines the network that a radio will attach to. This must be the same for all radios on your network.
JV	—	Enabled[1]	Verifies if a coordinator exists on the same channel to join the network or to leave if it cannot be found.
CE	Enabled[1]	—	Sets the device as coordinator.
NI	SENDER	RECEIVER	Defines the node identifier, a human-friendly name for the module. ⚠ The default **NI** value is a blank space. Make sure to delete the space when you change the value.
AP	API Enabled [1]	API Enabled [1]	Enables API mode.

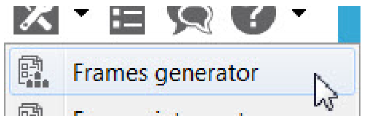
Step 2: Open the XCTU console

1. Switch to the **Consoles working mode** .
2. Open the serial connection with the radio module .
3. Change to the console of the other XBee module.
4. Open the serial connection with the radio module .
5. 

Step 3: Generate the Transmit Request frame

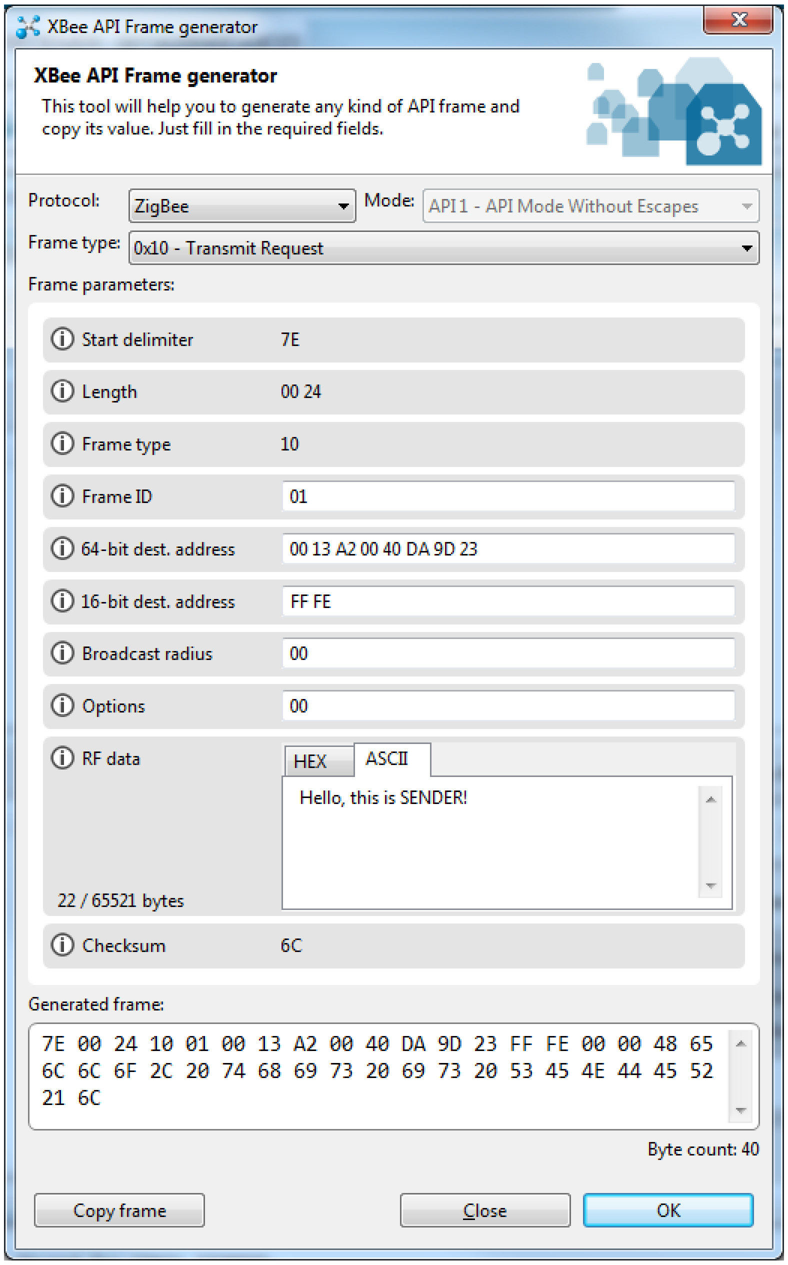
This topic describes how to generate a Transmit Request frame using the XCTU SENDER console.

1. Go to the **SENDER** console and detach it Detach to see two consoles at the same time.
2. In the SENDER console, click **Add new packet to the list** .
3. Open the **Frames Generator** tool.



4. In the **Protocol control**, select **Zigbee**.
5. In the **Frame type** control, select **0x10 - Transmit Request**.
6. In the **64-bit dest. address** box, type the 64-bit address of the RECEIVER module.
7. In the **RF data** box, click the **ASCII** tab and type the message "Hello, this is SENDER!"
8. Click **OK**.

9. Click **Add frame**.


XBee API Frame generator

XBee API Frame generator

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

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

Frame type: 0x10 - Transmit Request

Frame parameters:

Start delimiter	7E
Length	00 24
Frame type	10
Frame ID	01
64-bit dest. address	00 13 A2 00 40 DA 9D 23
16-bit dest. address	FF FE
Broadcast radius	00
Options	00
RF data (22 / 65521 bytes)	Hello, this is SENDER!
Checksum	6C

Generated frame:

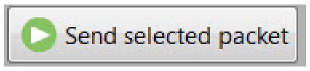
```
7E 00 24 10 01 00 13 A2 00 40 DA 9D 23 FF FE 00 00 48 65
6C 6C 6F 2C 20 74 68 69 73 20 69 73 20 53 45 4E 44 45 52
21 6C
```

Byte count: 40

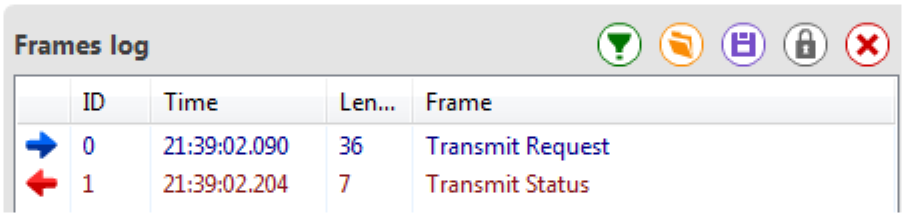
Step 4: Send the Transmit Request frame

After you have created a Transmit Request frame, you must send it.

1. Select the frame in the XCTU **Send frames** section.
2. Click **Send selected packet**.

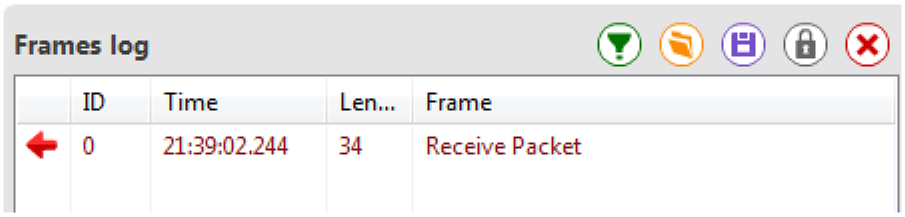


The **Frames log** indicates that one frame has been sent (blue) and another has been received (red).



	ID	Time	Len...	Frame
➡	0	21:39:02.090	36	Transmit Request
⬅	1	21:39:02.204	7	Transmit Status

Additionally, the RECEIVER console indicates that another packet has been received.

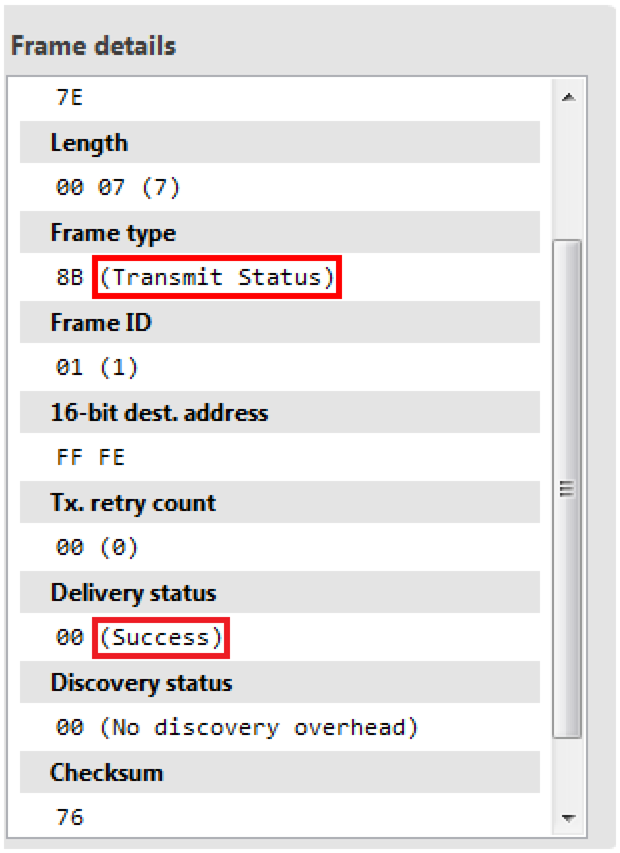


	ID	Time	Len...	Frame
⬅	0	21:39:02.244	34	Receive Packet

Step 5: Analyze the responses

Once you have sent the frames, you can analyze the responses on the receiving end.

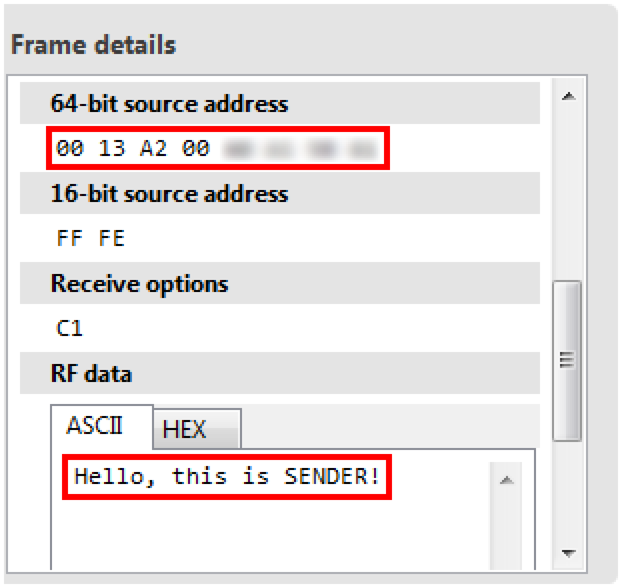
1. Select the received frame (Transmit Status) in the SENDER console to view the frame details on the right panel. Verify that the message was sent successfully.
   - **Frame type**: The received frame is a Transmit Status.
   - **Frame ID**: Since both frames have the same Frame ID, this is the response for the Transmit Request frame.
   - **Status**: The Success status indicates that the message was sent successfully.



Frame details

```
7E
Length
00 07 (7)
Frame type
8B (Transmit Status)
Frame ID
01 (1)
16-bit dest. address
FF FE
Tx. retry count
00 (0)
Delivery status
00 (Success)
Discovery status
00 (No discovery overhead)
Checksum
76
```

2. Analyze the details of the Receive Packet for RECEIVER. Verify that the message is the one you typed and the sender's address belongs to SENDER.
   - **Frame type**: The received frame is a Receive Packet
   - **64-bit source address**: This field displays the 64-bit address of the sender module, SENDER.
   - **Receive options**:
     - The packet was acknowledge (**0xC1** = **1100 0001**).
   - **RF data**: The message of the packet is "Hello, this is SENDER!".


Frame details

64-bit source address

```
00 13 A2 00 [redacted]
```

16-bit source address

```
FF FE
```

Receive options

```
C1
```

RF data

```
Hello, this is SENDER!
```

3. Disconnect both consoles by clicking **Close the serial connection** .

Libraries

Let's say you want to write an application to enable an intelligent device to monitor and manage an XBee network. You can write your own code to work with API mode, and you can also take advantage of existing software libraries that already parse the API frames. Depending on your preferred programming language and the intelligent device connected to the serial interface of the XBee, you can choose from a variety of available libraries:

- **XBee mbed Library** is a ready-to-import mbed extension to develop XBee projects on the mbed platforms. For more information, go to https://developer.mbed.org/teams/Digi-International-Inc/code/XBeeLib/.
- **Digi XBee Ansi C Library** is a collection of portable ANSI C code for communicating with XBee modules in API mode. For more information, go to https://github.com/digidotcom/xbee_ansic_library/.
- **XBee-arduino** is an Arduino library for communicating with XBees in API mode. For more information, go to https://code.google.com/p/xbee-ard uino/.
- **XBee Java Library** is an easy-to-use library developed in Java that allows you to interact with XBee modules working in API mode. For more information, visit the XBee Java Library documentation.

In this kit, you use the XBee Java Library to learn about the XBee features and capabilities offered in API operating mode. You can create several Java applications to control and monitor XBees connected to your computer via the XBee Grove Development Board.

Zigbee Mesh Network Setup

A Zigbee mesh network is created by a coordinator. Once the network has been created, other nodes can join it. The default device type of the XBee modules is router, so you must configure an XBee to be the coordinator. Note that the rest of the modules should be switched on (or reset) once the network is created so that they can join it properly.

Configure the device type of an XBee module

The device type of an XBee is determined by the value of two parameters: Coordinator Enable (**CE**) and Sleep Mode (**SM**). The first setting determines if an XBee module is coordinator or not, and the second one determines if the module is router or end device. Coordinators and routers cannot sleep, so the value for that setting must be always 0 (disabled).

	Configuration	Description

Coordinator	CE = 1
	SM = 0
		An XBee module is a coordinator if the CE setting is set to 1. When CE = 1, the value of the SM setting cannot be different than 0.

Router	CE = 0
	SM = 0
		An XBee module is a router if the CE setting is set to 0 and the sleep mode is disabled.

End device	CE = 0
	SM = 1
		An XBee module is an end device if it has any sleep mode enabled.

Startup operations

When you power on an XBee module, it performs several operations depending on the role assigned. The following sections explain the operations and commands performed by coordinators, routers, and end devices to form or join a network.

Coordinator

The coordinator is the only device that can start a network, so each Zigbee network must have one coordinator. It is responsible for selecting an unused operating channel, PAN ID, security policy, and stack profile for a network. To ensure the coordinator starts on a good channel and unused PAN ID, it performs a series of scans to discover any RF activity on different channels (energy scan) and to discover any nearby operating PANs (active scan).

The following commands control the coordinator network formation process:

- PAN ID (**ID**). Determines the PAN ID. If set to 0 (default), the device selects a random PAN ID.
- Scan Channels (**SC**). Determines the scan channels bitmask the coordinator uses to form a network. The coordinator performs an energy scan on all enabled SC channels.
- Scan Duration (**SD**). Sets the scan duration, which determines how long the coordinator performs an energy or active scan on a given channel.

After the coordinator has started the network, it can allow new devices to join it (up to 20 devices). The permit joining attribute is configurable with the Node Join Time (**NJ**) command. The coordinator can also route data packets and communicate with other devices on the network.

Note You can configure it to always allow joining (**FF**) for up to 254 seconds. However, Digi discourages this due to the security risk.

Router

Routers must discover and join a valid Zigbee network before they can participate in it. To discover nearby networks, the router performs an active scan, just like the coordinator does when it starts the network. When a router joins a network, it receives a randomly selected 16-bit address from the device that allowed the join .

Once a router joins a Zigbee network, it remains connected to the network on the same channel and PAN ID as long as it is not forced to leave. If the scan channels, PAN ID, and security settings do not change after a power cycle, it remains connected to the network after a power cycle. There are two provisions to automatically detect the presence of a network and leave if the check fails:

- Join Verification (**JV**). If enabled, the XBee attempts to discover the address of the coordinator when it first joins a network.
- Network Watchdog Timeout (**NW**). Used for a powered router to periodically check for the presence of a coordinator to verify network connectivity.

After a router has joined a network, it can allow new devices to join the network (up to 20 devices each router) with the Node Join Time (NJ) setting. It can also route data packets and communicate with other devices on the network.

End device

Similar to routers, end devices must also discover and join a valid Zigbee network before they can participate in it. End devices also discover networks by issuing an active scan, and when they join a network they receive a randomly selected 16-bit address from the device that allowed the join.

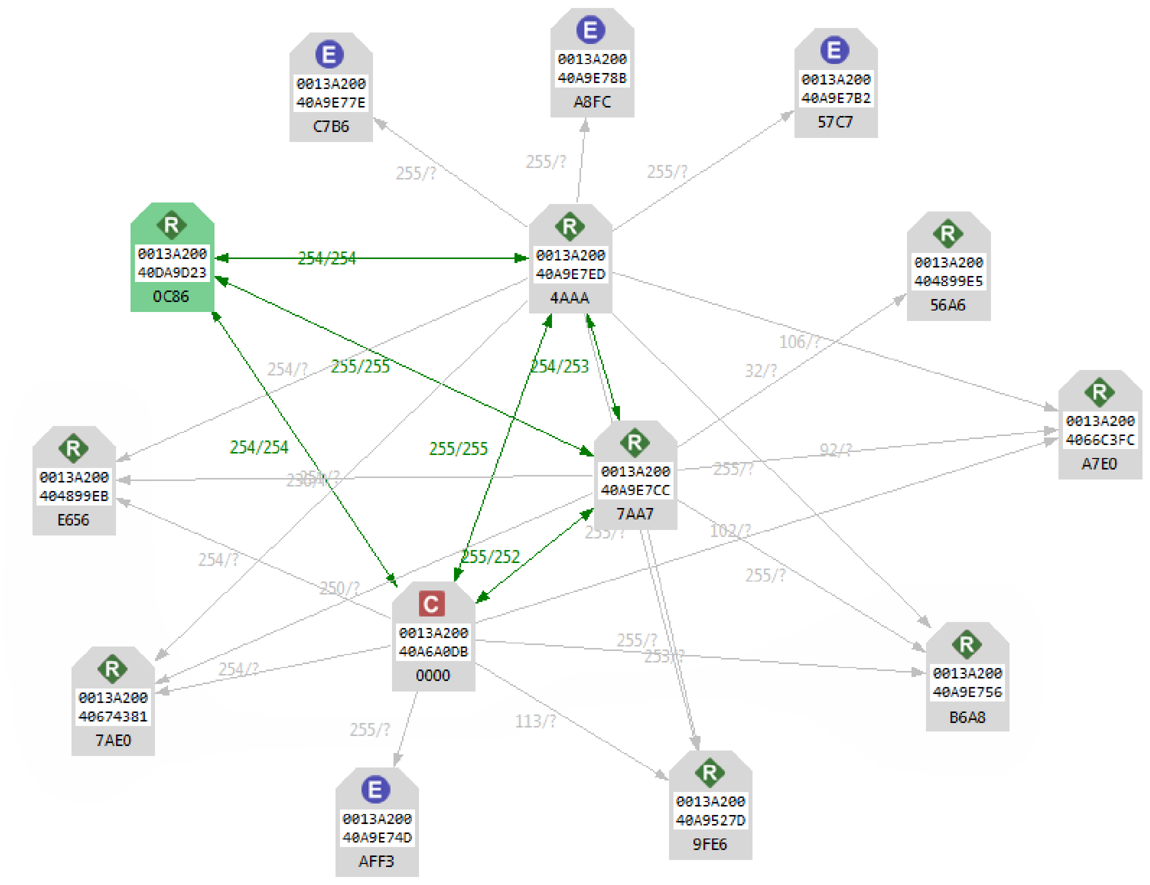
Since an end device may enter low power sleep modes and not be immediately responsive, it relies on the device that allowed the join to receive and buffer incoming messages on its behalf until it is able to wake and receive those messages. The device that allowed an end device to join becomes the parent, and the end device becomes the child. The end device polls its parent when it is awake to query for any new received data packets.

Coordinators and routers maintain a table of all child devices that have joined. This table has a finite size and determines how many end devices can join. You can use the Number of Remaining Children (**NC**) setting to determine how many additional end devices can join a coordinator or router.

After an end device has joined a network, it can communicate with other devices on that network. Since end devices are intended to be battery powered and therefore support low power (sleep) modes, they cannot allow other devices to join, nor can they route data packets.

Explore the network

To better understand how a Zigbee mesh network is formed, you can use XCTU's Network view to discover and visualize the topology and interconnections of the network.



To learn more about the Network View, see How-to: Visualize your network.

Section summary

To form a Zigbee mesh network, you must configure at least the following settings:

- PAN ID (**ID**). Every node should have the same value.
- Scan Channels (**SC**). Every node should have the same value.
- Channel Verification (**JV**). Enable this setting to ensure there is a coordinator in the network.
- Coordinator Enable (**CE**). Enable this setting in one module.
- Sleep Mode (**SM**). If you want to have end devices, set their sleep mode to some other value than 0.

Note There are other less critical settings under the Networking group in XCTU that also influence network creation.

Wireless data transmission

This section explains data transmission and guides you through an example to illustrate how it works. If you get stuck, see Troubleshooting.

Transmission methods

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

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

Broadcast transmission

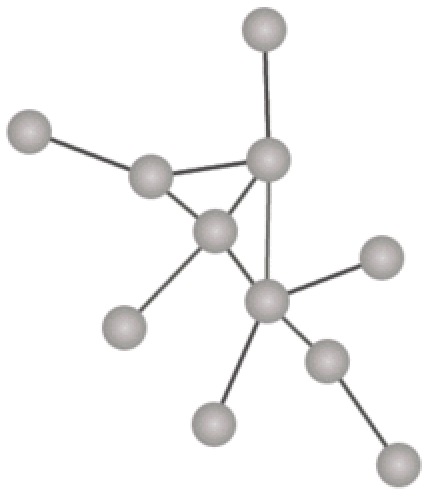
Broadcast means to transmit the same data to all nodes on a network. These transmissions are propagated throughout the entire network so that all possible nodes receive the transmission.

To accomplish this, the coordinator and all routers that receive a broadcast transmission re-transmit the data three times. When a router or coordinator delivers a broadcast transmission to an end device child, it sends the transmission only once, immediately after the end device wakes and polls the parent for new data.

You can address broadcast transmissions using either the 64-bit broadcast address or the 16-bit broadcast address:

- If the **64-bit broadcast address** (000000000000FFFF) is used, set the 16-bit address to unknown address (FFFE).
- If the **16-bit broadcast address** (FFFF) form is used, set the 64-bit address to unknown address (FFFFFFFFFFFFFFFF).

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



Unicast transmission

A **unicast** transmission consists of sending messages to a single node on the network identified by a unique address. The destination XBee can be an immediate neighbor of the sender, or be several hops away.

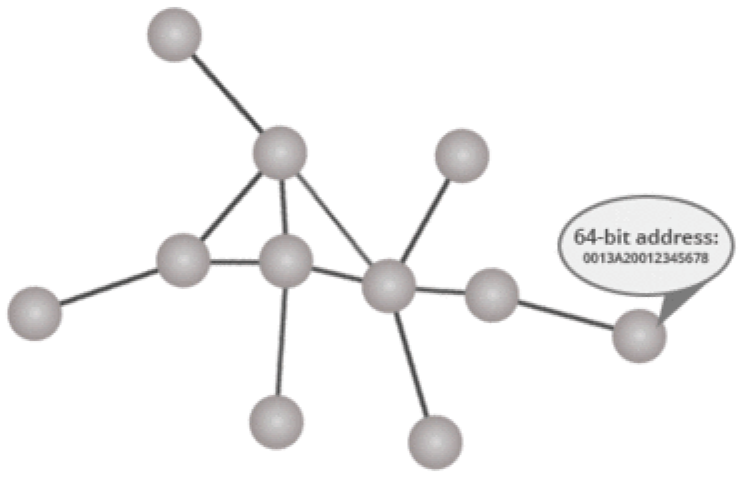
Wireless data may be addressed using either the 64-bit address or the 16-bit address (network address):

- If you use the **64-bit address**, set the network address to unknown address (FFFE).
- If you use the **16-bit address**, set the 64-bit address must be set to unknown address (FFFFFFFFFFFFFFFF).
- The 16-bit address 0000 and the 64-bit address 0000000000000000 are reserved for the coordinator.

The Zigbee network layer uses the 16-bit address of the destination on each hop to route the data.

If you use an invalid 16-bit address as a destination address, and the 64-bit address is unknown (FFFFFFFFFFFFFFFF), the Transmit Status (0x8B) message shows a delivery status code of 0x21 (network ACK failure) and a discovery status of 0x00 (no discovery overhead).

If you use a non-existent 64-bit address as a destination address, and the 16-bit address is unknown (FFFE), the device attempts address discovery and the Transmit Status (0x8B) message shows a delivery status code of 0x24 (address not found) and a discovery status code of 0x01 (address discovery attempted).



Address table

XBee devices use the destination network address to send data in a unicast transmission. Since data can only be sent using the destination's 64-bit address, all Zigbee devices maintain an **address table** to map 64-bit address to the corresponding 16-bit address. XBee modules can store up to 10 address table entries.

If the destination's 16-bit address is unknown:

1. The Zigbee stack uses its address table to look for an entry with a matching 64-bit address which determines the destination's 16-bit address.
2. If it is not found, the XBee automatically initiates a discovery process to find that address before transmitting the data:
   a. First, the sending device broadcasts an address discovery message. This message includes the 64-bit address of the remote XBee module whose 16-bit address is being requested.
   b. All nodes that receive this transmission compare their own 64-bit address to the one included in the message.
   c. If the addresses match, the remote XBee sends a response back to the requester module. This response includes the remote device's 16-bit address.
   d. When the requesting module receives this discovery response with the destination's 16-bit address, it transmits the data.

Example: transmit data

In the first example of this kit, you transmitted data to other nodes using transparent mode. An XBee module in transparent mode simply passes information along exactly as it receives it. This mode is an easy way to get started with XBees, but it has several limitations:

- You have to configure the address of the receiver in the transmitter module. If you want to transmit data to other XBees, you must re-configure the sender.
- The receiver module does not know who sent the message.

You can use API mode to avoid these limitations and have more flexibility and reliability in your data transmissions. In API mode, you still send the message to the module. But, you also send other necessary information, such as the destination address or checksum value, all wrapped in a packet with a defined structure called an API frame. This means that in API mode you don't need to set the destination address (DH + DL) in the module. Similarly, the receiver module receives more information than the message itself, such as the source address, signal strength, or checksum value.

In this example, you will create an application in the Java programming language to transmit data between the nodes of the network. To simplify that application, you will use the XBee Java Library, an easy-to-use API that allows you to interact with XBee modules.

Once you have everything set up, send a message to a specific device (unicast) or to all devices of the network (broadcast).

Several steps contain videos to help you successfully complete the example.

If you get stuck, see Troubleshooting.

Step 1: Requirements

For this setup you need the following hardware and software.

Hardware

- Three XBee Zigbee Mesh Kit modules
- Three XBee Grove Development Boards
- Three micro USB cables
- One computer

Software

- XCTU 6.3.1 or later
- XBee Java Library (XBJL-X.Y.Z.zip release file)
- Java Virtual Machine 6 or later
- A Java IDE (such as Eclipse or NetBeans)

Tip For more information about XCTU, see the XCTU walkthrough.

Step 2: Connect the components

To get started, connect the components and start XCTU.

1. Plug the XBee modules into the XBee Grove Development Boards and connect them to your computer using the micro USB cables provided. You can find more specific steps in Plug in the XBee module.
2. After connecting the modules to your computer, open XCTU.
3. Make sure you are in **Configuration working mode**.

Step 3: Configure the Xbee modules

Configure each of the three XBee modules to work in API mode and assign each a different role. As mentioned before, in this case you don't need to configure the destination address (DH + DL) of the modules.

1. Restore the default settings of all XBees with the **Load default firmware settings** button at the top of the Radio Configuration section.
2. Use XCTU to configure the following parameters:

Param	XBee A	XBee B	XBee C	Effect
ID	2015	2015	2015	Defines the network a radio will connect to. This parameter must be the same for all radios on your network.
JV	—	Enabled [1]	Enabled [1]	Verifies if a coordinator exists on the same channel to join the network or to leave if it cannot be found.
CE	Enabled [1]	—	—	Sets the device as coordinator.
NI	COORD	ROUTER	END_ DEVICE	Defines the node identifier, a human-friendly name for the module. The default NI value is a blank space. Make sure to delete the space when you change the value.
AP	API enabled [1]	API enabled [1]	API enabled [1]	Enables API mode.
SP	1F4	1F4	1F4	Defines the duration of time spent sleeping. 1F4 (hexadecimal) = 500 (decimal) x 10 ms = 5 seconds.
SM	—	—	Cyclic sleep [4]	Enables the cyclic sleep mode in the end device.
SO	—	—	2	Keeps the module awake during the entire period.

3. Write the settings of all XBees with the Write radio settings button at the top of the Radio Configuration section.

Step 4: Create a Java project

Create an empty Java project using Eclipse or NetBeans with the following project name: **XBeeTransmitDataCoord**.

Option 1: Eclipse

a. Select **File > New**, and click the **Java Project**.
b. The **New Java Project** window appears. Enter the Project name.
c. Click **Next**.

or

Option 2: NetBeans

a. Select **File > New project....**
b. The New Project window appears. In the Categories frame, select **Java > Java Application** from the panel on the right, and click **Next**.
c. Enter the Project name and the Project Location. Clear the Create Main Class option; you will create this later.
d. Click **Finish** to create the project. The window closes and the project appears in the Projects view list on the left side of the IDE.

Step 5: Link libraries to the project

This topic describes how to link the **XBee Java Library**, the **RXTX library** (including the native one), and the **logger library** to the project.

1. Download the XBJL_X.Y.Z.zip library.
2. Unzip the XBJL_X.Y.Z.zip library.
3. Link the libraries using Eclipse or NetBeans:

Option 1: Eclipse

a. Go to the **Libraries** tab of the New Java Project window.
b. Click **Add External JARs....**
c. In the JAR Selection window, search the folder where you unzipped the XBee Java Library and open the **xbee-java-library-X.Y.Z.jar** file.
d. Click **Add External JARs...** again.
e. Go to the **extra-libs** folder and select the following files:
   - **rxtx-2.2.jar**
   - **slf4j-api-x.y.z.jar**
   - **slf4j-nop-x.y.z.jar**
f. Expand the **rxtx-2.2.jar** file of the Libraries tab list, select **Native library location**, and click **Edit....**
g. Click **External folder...** to navigate to the **extra-libs\native\Windows\win32** folder of the directory where you unzipped the XBee Java Library file (XBJL_X.Y.Z.zip).
   - Replace **Windows\win32** with the directory that matches your operating system and the Java Virtual Machine installed (32 or 64 bits). If you don't know which Java Virtual Machine is installed in your computer, open a terminal or command prompt and

execute:

```
java -version
```

a. Click **OK** to add the path to the native libraries.
b. Click **Finish**.

or

Option 2: NetBeans

a. From **Projects** view, right-click your project and go to **Properties**.
b. In the categories list on the left, go to **Libraries** and click **Add JAR/Folder**.
c. In the **Add JAR/Folder** window, search the folder where you unzipped the XBee Java Library and open the **xbjlib-X.Y.X.ja**r file.
d. Click **Add JAR/Folder** again.
e. Go to the **extra-libs** folder and select the following files:
   - **rxtx-2.2.jar**
   - **slf4j-api- x.y.z .jar**
   - **slf4j-nop- x.y.z .jar**
f. Select **Run** in the left tree of the **Properties** dialog.
g. In the **VM Options** field, add the following option:

```
-Djava.library.path=<path_where_the_XBee_Java_Library_is_unzipped>\extra-
libs\native\Windows\win32
```

where:

   - **<path_where_the_XBee_Java_Library_is_unzipped>** is the absolute path of the directory where you unzipped the XBee Java Library file (XBJL_X.Y.Z.zip)
   - **Windows\win32** is the directory that matches your operating system and the Java Virtual Machine installed (32 or 64 bits). If you don't know which Java Virtual Machine is installed in your computer, open a terminal or command prompt and execute:

```
java -version
```

h. Click **OK**.

Step 6: Add the source code to the project

Follow these steps to add the source to the project.

1. Open the following source code, select all, and copy it to the clipboard: MainApp.
2. Add the Java source file with Eclipse or NetBeans.

Option 1: Eclipse

a. In the **Package Explorer** view, select the project and right-click.
b. From the context menu, select **New > Class**. The **New Java Class** wizard opens.
c. Type the **Name** of the class: **MainApp**.
d. Click **Finish**.

e. The **MainApp.java** file is automatically opened in the editor. Replace its contents with the source code you copied in the previous step.
f. A line at the top of the pasted code is underlined in red. Click on that line; a pop-up appears. Select the first option (**Move 'MainApp.java' to package '...'**) to resolve the error.

Option 2: NetBeans

a. In the **Projects** view, select the project and right-click.
b. From the context menu, select **New > Java Class...** The New Java Class wizard opens.
c. Modify the **Class Name** to be **MainApp**.
d. Click **Finish**.
e. The **MainApp.java** file automatically opens in the editor. Replace its contents with the source code you copied in the previous step.
f. A line at the top of the pasted code is underlined in red. Click on the light bulb next to that line; a pop-up appears. Select the first option (**Move class to correct folder**) to resolve the error.

Step 7: Set the port names and launch applications

For this step, set the port names for all three XBee modules, duplicate the project for the other XBee modules, then launch the applications.

1. Change the port name in the Java source code to match the port COORD is connected to.

```
// TODO: Replace with the port where your module is connected
private static final String PORT = "COM1";
// TODO: Replace with the baud rate of your module.
private static final int BAUD_RATE = 9600
```

2. Duplicate the Java project for the other XBee module and rename it to **XBeeTransmitDataRouter**.
3. Change the port name in the second project's source code to match the port ROUTER is connected to.
4. Duplicate the Java project for the last XBee module and rename it to **XBeeTransmitDataEndDevice**.
5. Change the port name in the third project's source code to match the port END_DEVICE is connected to.
6. Before launching each of the three applications, press the **Commissioning** button of the board the END_DEVICE is attached to wake it up. To identify the END_DEVICE module, look for the board where the On/Sleep LED is ON for 5 seconds and OFF for another 5 seconds.
7. Launch the three applications.

Step 8: Transmit data over the network

Follow these steps to transmit data to other XBee modules in your network with the unicast or broadcast method.

1. Send messages to a specific XBee module (unicast) or to all (broadcast) using the following pattern:

- **Unicast**: **NODE_IDENTIFIER: message**

  For example, to send the message "Hi XBee" to END_DEVICE:

  ```
  END_DEVICE: Hi XBee
  ```

- **Broadcast**: **ALL: message**

  For example, to send the message "Hi XBee nodes" to all nodes of the network:

  ```
  ALL: Hi XBee nodes
  ```

Step 9: Section summary of wireless data transmission

In this section, you have learned the following:

- An XBee sending a transmission in API mode not only transmits a raw message but also some extra information—such as the address of the source XBee module—packaged in what is called an API frame.
- In API mode, you don't need to set the DH and DL parameters of the receiver device because the destination address is already included in the API frame.
- API mode allows you to easily work with multiple destinations without needing to re-configure the sender module to establish a new destination module before sending the data.
- You can use the XBee Java Library to simplify and improve the use of the API operating mode.
- Depending on the number of devices that will receive the message, there are two types of transmissions:
  - Unicast sends a message to one node identified by a unique address.
  - Broadcast sends the same message to all possible nodes on the network.

Step 10: Do more with wireless data transmission

If you're ready to work more extensively with data transmission, try the following:

- Extend the network by adding more XBee Zigbee Mesh Kit modules so you can chat with other devices.

  Note To find the best channel to acquire more modules, see Where to buy XBee devices.

- Use Arduino or Raspberry Pi instead of a computer to transmit data wirelessly.

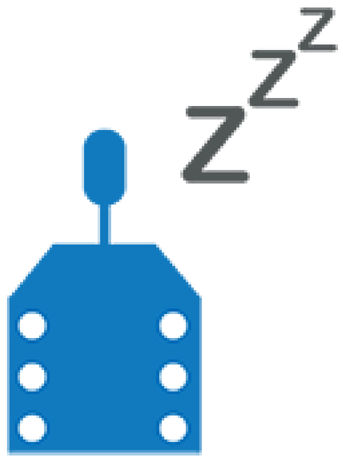
Low power and battery life

This section introduces the key concepts you need to know to take advantage of the power saving capabilities of XBee devices. It also provides a lab that lets you put the concepts to work and see the results.

Low power devices and battery life

The advantage of a wireless connection is that devices do not require physical wires to communicate, and they also use batteries instead of mains AC power. However, battery life can also be a major limitation. Depending on the location of the device, it can be difficult or expensive to replace the battery.

XBee modules are low-power devices. They can put themselves into a temporary sleep state in which they consume virtually no current. During sleep, the device is almost completely turned off and is sometimes incapable of sending or receiving data until it wakes up.



A real world scenario

Extending battery life is important in many real world scenarios. For example, if you had several greenhouses, each with a temperature sensor connected to an XBee device, battery life would be critical. Fully charged batteries would only power the modules for one day.

There are several ways to maximize battery life. For example:

- Putting the modules into a cycle where they sleep for one second and then wake for one second before sleeping again can double the battery life to two days.
- Cyclically sleep for 59 seconds and then waking for one second can keep the same batteries going for 60 days. Taking this further, you can potentially extend the battery life for years.

Design considerations for applications using sleep mode

Before using sleep mode you must take into consideration the structure of your project and your XBee network. Some applications, like the greenhouse example, are particularly suited to sleep mode. In that scenario, the modules only send data periodically and are not expected to receive data. The modules can therefore be sleeping most of the time and wake up only to send the temperature value.

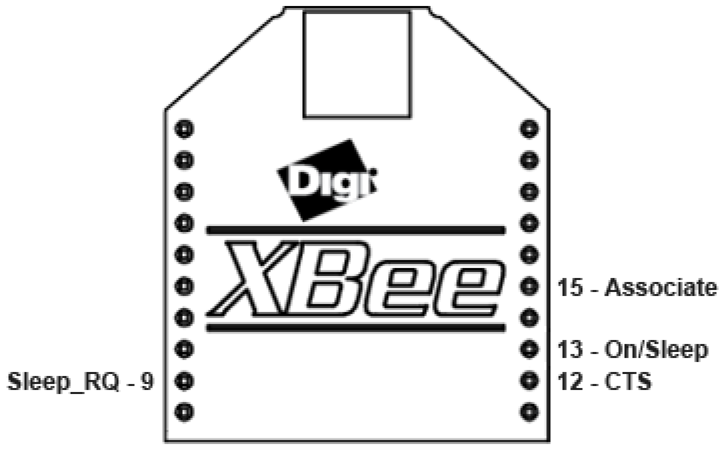
Sleep modes

XBee ZB end devices support three different sleep modes:

- Pin sleep (SM = 1)
- Cyclic sleep (SM = 4)
- Cyclic sleep with pin wake-up (SM = 5)

An end device in one of these sleep modes polls its parent every 100 milliseconds while it is awake to retrieve buffered data. When the module enters sleep mode:

- The module de-asserts (low) the On/Sleep pin (pin 13) to indicate the module is entering sleep mode.
- If CTS hardware flow control is enabled, the module de-asserts (high) the CTS pin (pin 12) to indicate that serial data should not be sent to the module.
- If the Associate pin (pin 15) is configured, it is driven low to avoid using power to light the LED.
- The Sleep_RQ pin (pin 9) is configured as a pulled-down input so that an external device can drive it high to wake the module (only applies to SM = 1 or SM = 5).
- The module leaves all other pins unmodified during sleep so they can operate as previously configured by the user.



When the XBee wakes from sleep:

- The device asserts (high) On/Sleep pin to indicate the it is awake.
- If you enable CTS hardware flow control, the CTS pin is asserted (low) indicating that serial data can be sent to the module.
- The Associate pin resumes its former configured operation.
- All other pins are left unmodified so they can operate as previously configured by the user.

Pin sleep

Pin sleep allows an external microcontroller to determine when the XBee should sleep and when it should wake by controlling the Sleep_RQ pin (pin 9). When Sleep_RQ is asserted (high) by connecting it to 3.3 volts, the module finishes any operation and enters a low power state. The module wakes when the Sleep_RQ pin is de-asserted (low).

Enable pin sleep mode by setting the Sleep Mode (**SM**) parameter to Pin Hibernate [1].

Cyclic sleep

Cyclic sleep allows the module to sleep for a specified time and wake for a short time to poll its parent for any buffered data messages before returning to sleep again.

Enable cyclic sleep mode by setting the Sleep Mode (**SM**) parameter to 4 or 5. The cyclic sleep with pin wake up (SM = 5) is a slight variation of the cyclic sleep mode (SM = 4) that allows the module to be woken prematurely by de-asserting the Sleep_RQ pin.

The following parameters control cyclic sleep:

Parameter	Name	Description
SP	Cyclic Sleep Period	Configures the sleep period of the module.
SN	Number of Cyclic Sleep Periods	Configures the number of sleep periods multiplier.
ST	Time before Sleep	Defines the period of inactivity of the module (during which no data is sent or received) before returning to cyclic sleep. If the XBee is transmitting or receiving a message, it will not go to sleep.
SO	Sleep Options	Defines options for sleep mode behavior: 0x02: Always wake for full ST time. 0x04: Enable extended sleep (sleep for full SP * SN time).

Press the **Commissioning** button to wake a sleeping device for 30 seconds.

Example: enable sleep mode

This example shows you how to extend the battery life of an XBee Zigbee module. The example uses all three modules included in the kit to demonstrate how a Zigbee network handles messages when some modules are sleeping.

Configure one of the modules as coordinator and the other two as end devices with different sleep modes. An end device periodically sends the value of an ADC to the other end device. Since the receiver is asleep, the coordinator stores all of its messages and forwards them to the destination module once it wakes up.

Tip If you get stuck, see Troubleshooting.

Step 1: Requirements

For this setup you need the following hardware and software.

Hardware

- Three XBee Zigbee Mesh Kit modules
- Three XBee Grove Development Boards

- Three micro USB cables
- One computer

Software

- XCTU 6.3.1 or later

Tip For more information about XCTU, see the XCTU walkthrough.

Step 2: Connect the components

To get started, connect the components and start XCTU.

1. Plug the XBee modules into the XBee Grove Development Boards and connect them to your computer using the micro USB cables provided. You can find more specific steps in Plug in the XBee module.
2. After connecting the modules to your computer, open XCTU.
3. Make sure you are in **Configuration working mode**.

Step 3: Configure the XBee Modules

To transmit data wirelessly between your XBees, you must configure them to be in the same network. Remember that in the Zigbee protocol, one device must be the coordinator and the rest can be routers or end devices. In this case, you will have two end devices configured to sleep using different sleep modes: one with cyclic sleep (XBee B) and the other with pin hibernate sleep (XBee C).

The coordinator is responsible for storing messages sent from XBee B to XBee C while XBee C is asleep. It forwards the messages once it wakes up.

1. Restore the default settings of all XBee modules with the **Load default firmware settings** button at the top of the Radio Configuration section.
2. Use XCTU to configure the following parameters:

Param	XBee A	XBee B	XBee C	Effect
ID	2015	2015	2015	Defines the network for a radio to attach to. This must be the same for all radios on your network.
JV	—	Enabled [1]	Enabled [1]	Verifies if a coordinator exists on the same channel to join the network or to leave if it cannot be found.
CE	Enabled [1]	—	—	Sets the device as coordinator.
DH	—	0013A200	—	Defines the destination address (high part) to transmit the data to.
DL	—	SL of XBee C	—	Defines the destination address (low part) to transmit the data to. Use the address 0000000000000000 to address the coordinator.

Param	XBee A	XBee B	XBee C	Effect
NI	COORD	ED_CYCLIC	ED_PIN	Defines the node identifier, a human-friendly name for the module. ⚠ The default **NI** value is a blank space. Make sure to delete the space when you change the value.
AP	API enabled [1]	API enabled [1]	API enabled [1]	Enables API operating mode.
SP	384	384	—	Defines the duration of time spent sleeping. 384 (hexadecimal) = 900 (decimal) x 10 ms = 9 seconds.
SM	—	Cyclic sleep [4]	Pin Hibernate [1]	Enables cyclic sleep mode on XBee B and pin hibernate sleep mode on XBee C (both end devices).
ST	—	7D0	—	Defines the period of inactivity (no serial or RF data received) before going to sleep. 7D0 (hexadecimal) = 2000 (decimal) x 1 ms = 2 seconds.
SO	—	2	—	Keeps the module awake during the entire period.

Param	XBee A	XBee B	XBee C	Effect
D2/D3	—	ADC [2]	—	Sets the DIO2/AD2 or DIO3/AD3 pin as ADC in XBee B, depending on if the XBee module is THT or SMT. This pin is connected to a potentiometer. ⚠ Configure the **D2** parameter as ADC [2] only if XBee B is surface-mount (SMT). However, if XBee B is through-hole (THT), you have to configure the **D3** par ameter as ADC [2] instead of the **D2**.
IR	—	3E8	—	Configures XBee B to send an IO sample every second (1000 ms = 3E8 in hexadecimal).

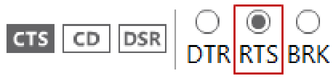
Note The dash (**—**) in the table means to keep the default value. Do not change the default value.

3. Write the settings of all XBee modules with the **Write radio settings** button at the top of the Radio Configuration section.

Step 4: Sleep

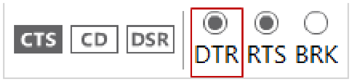
With this configuration, ED_CYCLIC sends the value of the potentiometer to ED_PIN every time it wakes up. The coordinator stores all the D_CYCLIC samples sent to ED_PIN until it wakes up. To verify, perform the following steps in XCTU:

1. Select ED_PIN module (receiver).
2. Switch to the **Consoles working mode**.
3. Open the serial connection with the module.
4. To request the module to sleep, click the **DTR** radio button from the top of the console to deactivate it (notice that the CTS indicator is also deactivated).

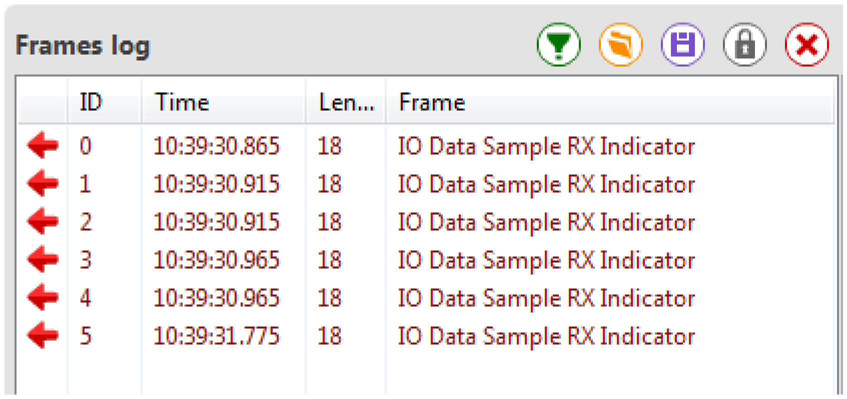


The **DTR** pin is the same as the **Sleep_RQ** pin. When the DTR option is deactivated, the module goes to sleep; when DTR is activated, the module wakes up.

5. After 20 seconds or so, activate the **DTR** button.



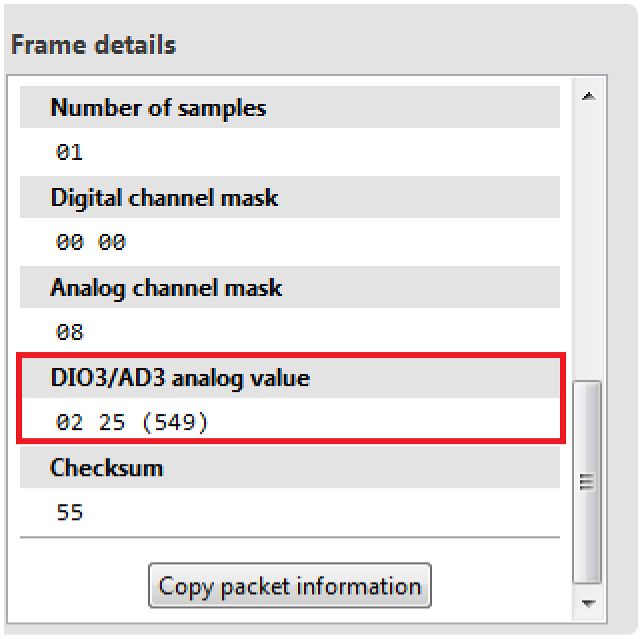
6. Check that the module receives a series of IO Samples (IO Data Sample RX Indicator).


Frames log

ID	Time	Len...	Frame
0	10:39:30.865	18	IO Data Sample RX Indicator
1	10:39:30.915	18	IO Data Sample RX Indicator
2	10:39:30.915	18	IO Data Sample RX Indicator
3	10:39:30.965	18	IO Data Sample RX Indicator
4	10:39:30.965	18	IO Data Sample RX Indicator
5	10:39:31.775	18	IO Data Sample RX Indicator

When the module wakes up, it immediately receives several IO samples instead of receiving one every second (IR parameter). This happens because the coordinator stores the samples that ED_PIN is not able to receive while it is asleep. Once ED_PIN wakes up, the coordinator sends all IO samples at once.

7. Select one frame and check its details in the right panel. The value of the potentiometer (DIO3_AD3) and other details related to the frame appear.



Number of samples
01
Digital channel mask
00 00
Analog channel mask
08
DIO3/AD3 analog value
02 25 (549)
Checksum
55

8. Repeat step 4 and rotate the potentiometer of the board where ED_CYCLIC is attached. Wait for another 20 seconds and check that the new packets that arrive contain a different value for the DIO3_AD3 pin.
9. If you leave the receiver module (ED_PIN) awake, ED_PIN receives an IO sample every second while the ED_CYCLIC is awake (ST val ue is two seconds). After that, ED_CYCLIC goes to sleep for nine seconds (SP parameter). The cycle then starts over again.

Note Make sure to close the serial connection with the module when you finish the example.

Step 5: What have you learned?

In this section, you have learned that:

- Modules with Zigbee protocol, as well as others, can go into a temporary sleep state in which they consume virtually no current. In Zigbee, only the modules configured as End Devices can go to sleep.
- When an end device is asleep its parent (the router or coordinator that allows the end device to join the network) buffers its data until a timeout expires (**SP**), or until the end device sends a poll request to retrieve the data.

- Pins 9 and 13 are related to the sleep modes. You can use pin 9 to put the module to sleep, and pin 13 to determine the sleep state of the device.
- While an XBee is in sleep mode, there is no data transmission or reception. If you try to communicate with the module when it is asleep, XCTU displays a warning message saying that the module must be reset to wake up.
- To configure your module to go to sleep, you must configure the following parameters:
  - Sleep Mode (**SM**):
    - Pin sleep mode (SM = 1) pull high pin 9 by connecting it to 3.3 volts to put the module to sleep. The module will wake up when pin 9 is de-asserted (low).
    - Cyclic sleep modes (SM = 4 and SM = 5) enable the module to sleep and wake up on a fixed schedule. These modes need the **ST** and **SP** parameters to be configured.
  - Time before Sleep (**ST**) is the period of time during which no data is sent or received (while the module is awake) before returning to cyclic sleep. This parameter is only applicable for cyclic sleep modes.
  - Cyclic Sleep Period (**SP**) is the length of time an XBee remains asleep. This parameter is only applicable for cyclic sleep modes.

Step 6: Extend the example

If you are ready to move beyond this exercise and extend the example, try the following:

- Connect a battery to ED_CYCLIC (XBee B), ED_PIN (XBee C), or both and move the modules away from COORD (XBee A).
- Combine this feature with a real sensor to create a low-power sensor network.

Inputs and outputs

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

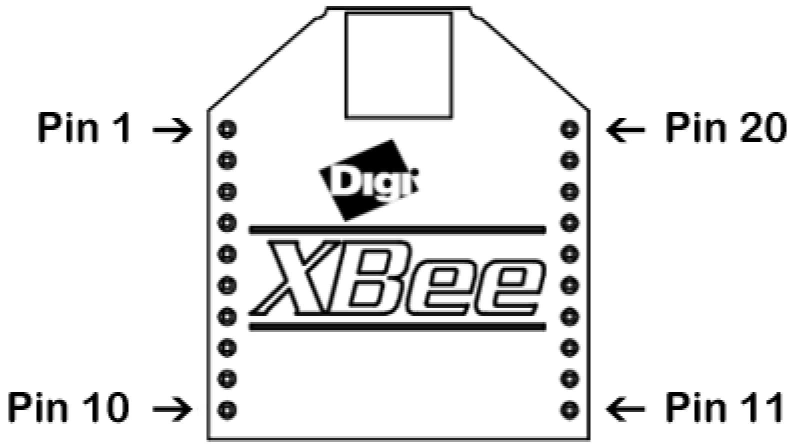
With these pins you can, for example, turn on a light by sending information to an XBee module connected to an actuator, or measure the outside temperature by obtaining data from a temperature sensor attached to your XBee module.

Learn about I/O pins, sensors, actuators in this section, then put your knowledge to work by using sensors.

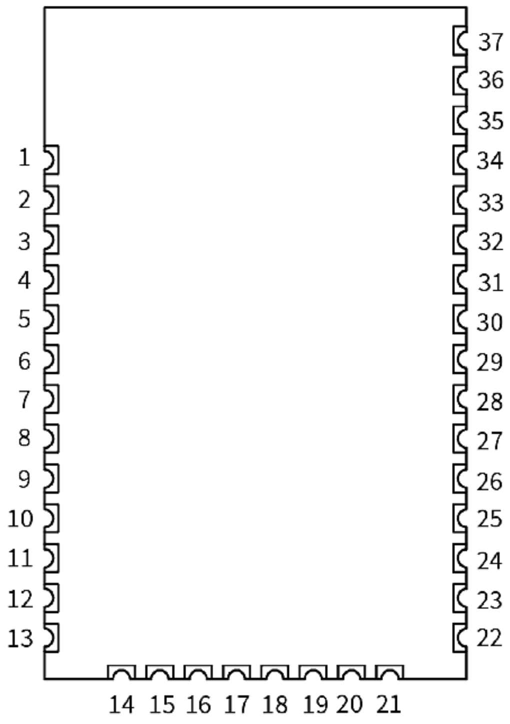
XBee I/O pins

The following table shows the I/O pins of the XBee THT and XBee SMT modules:

XBee THT Model



XBee SMT model



Pin name	Physical pin#		Parameter
	THT	SMT	
DIO0, AD0	20	33	D0
DIO1, AD1	19	32	D1
DIO2, AD2	18	31	D2
DIO3, AD3	17	30	D3
DIO4	11	24	D4
DIO5	15	28	D5
DIO6	16	29	D6
DIO7	12	25	D7
DIO8	9	10	D8
DIO9	13	26	D9
DIO10, PWM RSSI	6	7	P0
PWM1, DIO11	7	8	P1
DIO12, PWM2	4	21	P2
DIO13	2	3	P3
DIO14	3	4	P4
DIO15	—	17	P5
DIO16	—	16	P6
DIO17	—	15	P7
DIO18	—	14	P8
DIO19	—	12	P9

(D = digital, I = input, O = output, AD = analog input, PWM = pulse-width modulation, — not available)

Note The number and type of IOs available can vary between different module variants.

Sensors

A **sensor** is a device that detects events or changes and provides a corresponding output, generally as an electrical signal.

There are two types of sensors: digital and analog. A motion sensor is a digital sensor because it can return two discrete values: movement detected or movement not detected. Other digital sensors might provide a binary value. A digital compass, for example, may provide your current heading by sending a 9-bit value with a range from 0 to 359. On the other hand, a thermometer is an analog sensor because the voltage output changes gradually as the temperature changes.

Setting pins for digital and analog sensors

Configure the pin of your XBee module according to the sensor that is connected to it:

- If you connect a digital sensor, configure the pin as Digital Input.
- If you connect an analog sensor, configure the pin as Analog to Digital Converter (ADC).

Note For more information about sensors, see the How XBee devices get sensor data section.

Actuators

An actuator is a device that is responsible for controlling a mechanism or system. The XBee device offers some simple output functions so that basic actuations can take place. For example, you can send digital information directly to an XBee device and direct it to turn on a light or start up a motor.

Set pins for digital and analog actuators

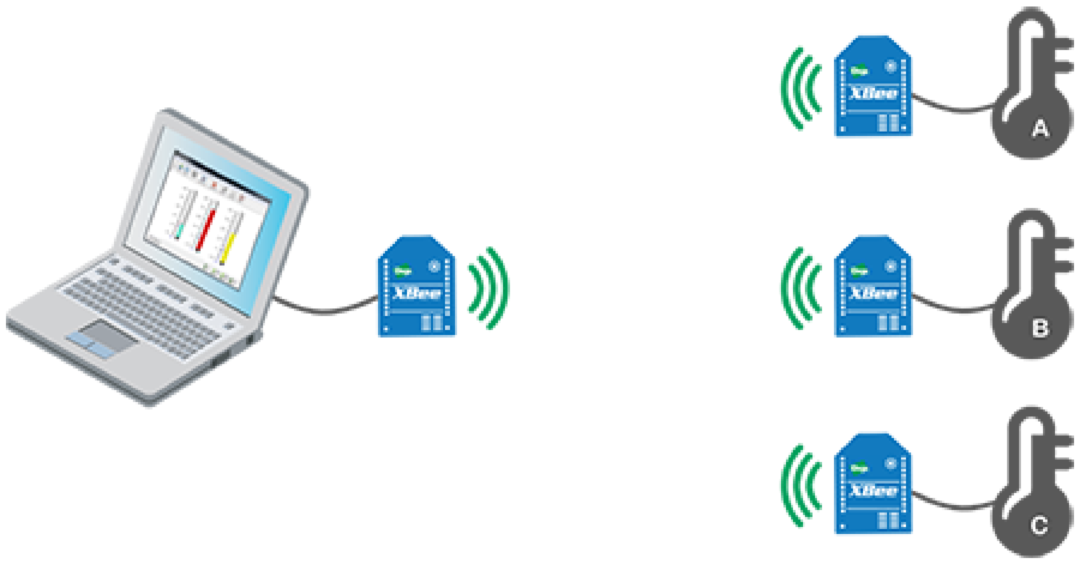
Configure the pin of your XBee device according to the actuator that is connected to it:

- If you connect a digital actuator, configure the pin as Digital Output.
- If you connect an analog actuator, configure the pin as PWM (analog output).

For more information about sensors, see How XBee modules control devices.

How XBee devices get sensor data

XBee devices are often used to form **sensor networks**. In a sensor network, the main device—also called the local XBee device—receives data from the sensors attached to the remote XBee devices.

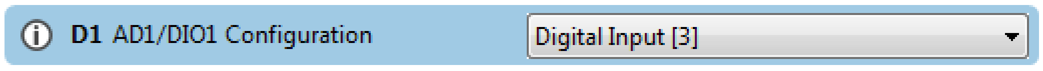


To receive that data, you must configure the remote XBee devices to "listen" on the particular pin where the sensor is connected and to send the data to the main XBee device.

How to configure a pin as an input

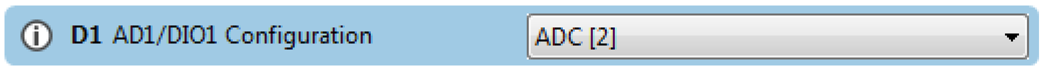
Configure a pin for digital input

You can configure a pin through XCTU. If your sensor reads digital values (like a doorbell) and is connected to the DIO1/AD1 pin, configure the D1 parameter as Digital Input [3]:


D1 AD1/DIO1 Configuration: Digital Input [3]

Configure a pin for analog input

If your sensor reads analog values (like a temperature sensor) and is connected to the DIO1/AD1 pin, configure the D1 parameter as ADC [2]:


D1 AD1/DIO1 Configuration: ADC [2]

How to obtain data from a sensor

There are two ways to obtain sensor information:

- Queried sampling to immediately read all enabled digital and analog input pins.
- Automatic sampling to transmit the sensor data periodically or whenever a digital pin changes.

In both cases, the information is sent to the other module is called **IO sample**. It contains which inputs (DIO lines or ADC channels) have sampling enabled and the value of all the enabled digital and analog inputs.

Queried sampling (IS)

The Force Sample (**IS**) command forces a read of all enabled digital and analog input pins. You can send it locally or to a remote device.

Use the XCTU console or any serial port terminal application to send this command.

When the module sends the **IS** command, the receiving device reads all enabled digital IO and analog input channels and returns their value. If the module transmits the **IS** command locally, it sends the IO data out the serial interface. If the module transmits the **IS** command to a remote XBee module, it sends the remote IO data over the air to the requester module.

Automatic sampling

Once you have set up the pin, the remote module must be configured to automatically transmit the sensor information to the main XBee module. The remote XBee module needs to know:

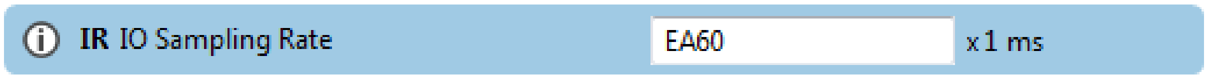
1. Where to transmit the sensor data: define this information for the module receiving this information by the destination address (**DH** + **DL**) parameters.
2. When to transmit the sensor data:
   - Periodically: The XBee can send the information read from the sensor at a specified interval.
   - By change detection: When a pin or several pins change status.

   Configure parameters IO Sampling Rate (**IR**) and Digital IO Change Detection (**IC**) to automatically transmit the sensor data.

Note These two features can work in combination with each other, depending on your requirements. For example, you could choose to receive an IO sample every minute (**IR**) but also when a certain pin changes state (**IC**).

IO Sampling Rate (IR)

The **IR** parameter sets the I/O sample rate: that is, how frequently to report the current pin state and transmit it to the destination address. The rate is set in milliseconds using hexadecimal notation. The value 0 disables the feature.


IR IO Sampling Rate EA60 x 1 ms

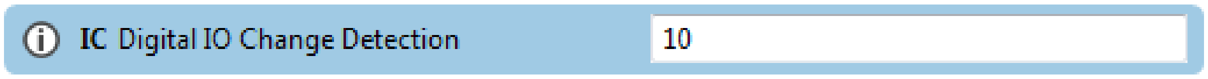
For example, if you want to transmit the sensor info every minute, set this parameter to EA60 (1 minute = 60 seconds = 60000 ms = EA60 hex).

Use XCTU to configure the sample rate interval.

Note Sleeping devices, configured to send samples periodically, transmit the first sample immediately after waking up, and then continue sending periodic IO samples at the **IR** rate, until the Time Before Sleep (**ST**) timer expires and the device can resume sleeping.

Digital IO Change Detection (IC)

The **IC** parameter allows you to set which pins to monitor for change detection. When the state of the monitored pin(s) changes, a sample is immediately sent to the destination address.


IC Digital IO Change Detection 10

Use XCTU to set the value of **IC** parameter.

To select which pins monitor, assign a binary value to **IC** parameter based on the following pattern:

DIO12	DIO11	DIO10	DIO9	DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1	DIO0
0	0	0	0	0	0	0	0	0	0	0	0	0

For example, if you want to monitor DIO1, the value would be **0000000000010**, which is 2 in hexadecimal notation. If you want to monitor DIO12, DIO8, DIO3 and DIO1, the value would be **1000100001010** (binary) = **110A** (hexadecimal). The value **0** disables the feature.

The Digital IO Change Detection (**IC**) feature only works for digital pins, so you will not receive anything if the value of an analog pin changes.

If an XBee module is sleeping, changes in any of the monitored pins will not wake the module.

Example: receive digital data

This section teaches you how to create an XBee digital sensor network. Since the kit does not contain a real sensor, you simulate a digital sensor with the user button of the XBee Grove Development Board.

Note If you have a Grove sensor, you can connect it to the board for a more realistic example of a sensor network.

You will configure one of the modules as coordinator and the others as router. The routers (where the sensors are connected) will be the senders and will transmit the status of the user button to the coordinator every time the button is pressed or released.

If you get stuck, see Troubleshooting.

Step 1: Requirements

For this setup you need the following hardware and software.

Hardware

- Three XBee Zigbee Mesh Kit modules
- Three XBee Grove Development Boards
- Three micro USB cables
- One computer

Software

- XCTU 6.3.1 or later
- XBee Java Library (XBJL-X.Y.Z.zip release file)
- Java Virtual Machine 6 or later
- A Java IDE (such as Eclipse or NetBeans)

Tip For more information about XCTU, see the XCTU walkthrough.

Step 2: Connect the components

If you have a Grove sensor (not included in the kit), plug it into the Grove DIO4 connector of the sender XBee's board. Follow the steps to connect your modules:

1. Plug the XBee modules into the XBee Grove Development Boards and connect them to your computer using the micro USB cables provided. You can find more specific steps in Plug in the XBee module.
2. After connecting the modules to your computer, open XCTU.
3. Make sure you are in **Configuration working mode**.

Step 3: Configure the XBee modules

XBee B and XBee C, the routers, send the digital value from the user button to XBee A, the coordinator, every time the value changes (that is, when you press or release the button).

Set the destination address (**DH** + **DL**) of the senders (XBee B and XBee C) to the MAC address (**SH** + **SL**) of the receiver (XBee A). Additionally, configure the pin where the button is connected (DIO4/AD4) as a digital input, and set the DIO change detect (IC) to monitor the same pin.

1. Restore the default settings of all XBee modules with the **Load default firmware settings** button at the top of the Radio Configuration section.
2. Use XCTU to configure the following parameters:

Param	XBee A	XBee B	XBee C	Effect
ID	2015	2015	2015	Defines the network that a radio will attach to. This must be the same for all radios in your network.
JV	—	Enabled [1]	Enabled [1]	Verifies if a coordinator exists on the same channel to join the network or to leave if it cannot be found.
CE	Enabled [1]	—	—	Sets the device as coordinator.
DH	—	0	0	Defines the destination address (high part) to transmit the data to. The address 0000000000000000 can be used to address the coordinator.
DL	—	0	0	Defines the destination address (low part) to transmit the data to. The address 0000000000000000 can be used to address the coordinator.
NI	COORD	ROUTER_1	ROUTER_2	Defines the node identifier, a human-friendly name for the module. ⚠ The default NI value is a blank space. Make sure to delete the space when you change the value.

Param	XBee A	XBee B	XBee C	Effect
AP	API enabled [1]	API enabled [1]	API enabled [1]	Enables the API operating mode.
D4	—	Digital Input [3]	Digital Input [3]	Sets the DIO4/AD4 pin as digital input in the senders. This pin is connected to a button.
IC	—	10	10	Configures the senders to transmit an IO sample when pin DIO4 (where the button is connected) changes. 00010000 (binary) = 10 (hexadecimal). For more information on how to configure this parameter to monitor the pins, see How to obtain data from a sensor.

Note The dash (**—**) in the table means to keep the default value. Do not change the default value.

3. Write the settings of all XBee modules with the **Write radio settings** button at the top of the Radio Configuration section.

Step 4: Create a Java project

Create an empty Java project named using Eclipse or NetBeans, with the following project name: **ReceiveDigitalData**.

Option 1: Eclipse

a. Select **File > New**, and click the **Java Project**.
b. The **New Java Project** window appears. Enter the Project name.
c. Click **Next**.

or

Option 2: NetBeans

a. Select **File > New project....**
b. The New Project window appears. In the Categories frame, select **Java > Java Application** from the panel on the right, and click **Next**.
c. Enter the Project name and the Project Location. Clear the Create Main Class option; you will create this later.
d. Click **Finish** to create the project. The window closes and the project appears in the Projects view list on the left side of the IDE.

Step 5: Link libraries to the project

This topic describes how to link the **XBee Java Library**, the **RXTX library** (including the native one), and the **logger library** to the project.

1. Download the XBJL_X.Y.Z.zip library.
2. Unzip the XBJL_X.Y.Z.zip library.
3. Link the libraries using Eclipse or NetBeans:

Option 1: Eclipse

a. Go to the **Libraries** tab of the New Java Project window.
b. Click **Add External JARs....**
c. In the JAR Selection window, search the folder where you unzipped the XBee Java Library and open the **xbee-java-library-X.Y.Z.jar** file.
d. Click **Add External JARs...** again.
e. Go to the **extra-libs** folder and select the following files:
   - **rxtx-2.2.jar**
   - **slf4j-api-x.y.z.jar**
   - **slf4j-nop-x.y.z.jar**
f. Expand the **rxtx-2.2.jar** file of the Libraries tab list, select **Native library location**, and click **Edit....**
g. Click **External folder...** to navigate to the **extra-libs\native\Windows\win32** folder of the directory where you unzipped the XBee Java Library file (XBJL_X.Y.Z.zip).
   - Replace **Windows\win32** with the directory that matches your operating system and the Java Virtual Machine installed (32 or 64 bits). If you don't know which Java Virtual

Machine is installed in your computer, open a terminal or command prompt and execute:

```
java -version
```

h. Click **OK** to add the path to the native libraries.
i. Click **Finish**.

or

Option 2: NetBeans

a. From **Projects** view, right-click your project and go to **Properties**.
b. In the categories list on the left, go to **Libraries** and click **Add JAR/Folder**.
c. In the **Add JAR/Folder** window, search the folder where you unzipped the XBee Java Library and open the **xbjlib-X.Y.X.ja**r file.
d. Click **Add JAR/Folder** again.
e. Go to the **extra-libs** folder and select the following files:
   - **rxtx-2.2.jar**
   - **slf4j-api- x.y.z .jar**
   - **slf4j-nop- x.y.z .jar**
f. Select **Run** in the left tree of the **Properties** dialog.
g. In the **VM Options** field, add the following option:

```
-Djava.library.path=<path_where_the_XBee_Java_Library_is_unzipped>\extra-
libs\native\Windows\win32
```

where:

- **<path_where_the_XBee_Java_Library_is_unzipped>** is the absolute path of the directory where you unzipped the XBee Java Library file (XBJL_X.Y.Z.zip)
- **Windows\win32** is the directory that matches your operating system and the Java Virtual Machine installed (32 or 64 bits). If you don't know which Java Virtual Machine is installed in your computer, open a terminal or command prompt and execute:

```
java -version
```

h. Click **OK**.

Step 6: Add the source code to the project

Follow these steps to add the source to the project.

1. Open the following source code, select all, and copy it to the clipboard: MainApp.java.
2. Add the Java source file with Eclipse or NetBeans.

Option 1: Eclipse

a. In the **Package Explorer** view, select the project and right-click.
b. From the context menu, select **New > Class**. The **New Java Class** wizard opens.
c. Type the **Name** of the class: **MainApp**.
d. Click **Finish**.

e. The **MainApp.java** file is automatically opened in the editor. Replace its contents with the source code you copied in the previous step.
f. A line at the top of the pasted code is underlined in red. Click on that line; a pop-up appears. Select the first option (**Move 'MainApp.java' to package '...'**) to resolve the error.

Option 2: NetBeans

a. In the **Projects** view, select the project and right-click.
b. From the context menu, select **New > Java Class...** The New Java Class wizard opens.
c. Modify the **Class Name** to be **MainApp**.
d. Click **Finish**.
e. The **MainApp.java** file automatically opens in the editor. Replace its contents with the source code you copied in the previous step.
f. A line at the top of the pasted code is underlined in red. Click on the light bulb next to that line; a pop-up appears. Select the first option (**Move class to correct folder**) to resolve the error.

Option 1: Eclipse

a. In the **Package Explorer** view, select the project and right-click.
b. From the context menu, select **New > Class**. The **New Java Class** wizard opens.
c. Type the **Name** of the class: **MainApp**.
d. Click **Finish**.
e. The **MainApp.java** file is automatically opened in the editor. Replace its contents with the source code you copied in the previous step.
f. A line at the top of the pasted code is underlined in red. Click on that line; a pop-up appears. Select the first option (**Move 'MainApp.java' to package '...'**) to resolve the error.

Option 2: NetBeans

a. In the **Projects** view, select the project and right-click.
b. From the context menu, select **New > Java Class...** The New Java Class wizard opens.
c. Modify the **Class Name** to be **MainApp**.
d. Click **Finish**.
e. The **MainApp.java** file automatically opens in the editor. Replace its contents with the source code you copied in the previous step.
f. A line at the top of the pasted code is underlined in red. Click on the light bulb next to that line; a pop-up appears. Select the first option (**Move class to correct folder**) to resolve the error.

Step 7: Set the port name and launch the application

For this step, set the port name and launch the application.

1. Change the port name in the Java source code to match the port that the COORD module (receiver) is connected to.

```
// TODO: Replace with the port where your coordinator module is connected
private static final String PORT = "COM1";
```

```
// TODO: Replace with the baud rate of your coordinator module.
private static final int BAUD_RATE = 9600
```

2. Launch the application on your computer. Every time you press or release the ROUTER_1 (XBee B) or ROUTER_2 (XBee C) user button, COORD (XBee A) receives its status.
3. Press the button and check the received status. The output of the application is similar to the following:

```
+----------------------------------------+
|    Receive Digital Data Sample         |
+----------------------------------------+

WARNING: RXTX Version mismatch
       Jar version = RXTX-2.2pre1
       native lib Version = RXTX-2.2pre2

Listening for IO samples... Press the user button of any remote device.

Digital data from '0013A20012345678': Low (button pressed)
Digital data from '0013A20012345678': High (button released)
Digital data from '0013A20012345678': Low (button pressed)
Digital data from '0013A20012345678': High (button released)
```

Step 8: Section summary of receiving digital data

In this section, you have learned that:

- All XBee modules have a set of pins you can use to connect and configure sensors or actuators.
- A sensor is a device that provides a corresponding output as a response to events or changes in quantities. There are two types:
  - Digital sensors return discrete values such as on/off.
  - Analog sensors can return a wide variety of values such as the temperature of a room.
- If, as in this example, you want to read data from a digital sensor, you must configure the selected IO as Digital Input.
- You can obtain digital data from a sensor by configuring the remote XBee to transmit the IO data:
  - To the local device by setting the **DH** and **DL** parameters to the MAC of the receiver module.
  - When a digital pin changes (using the **IC** or Digital IO Change Detection parameter) as you did in this example, or periodically (using the **IR** or IO Sampling Rate parameter).
- Although the Digital IO Change Detection (**IC**) parameter is configured to monitor one or more pins, changes in these pins do not wake an end device while it is sleeping.
- The data sent from one module to the other is called IO Sample. It contains the inputs (DIO lines or ADC channels) for which sampling has been enabled. It also contains the value of all enabled digital and analog inputs.

Step 9: Do more with receiving digital data

If you are ready to work more extensively with receiving digital data, try the following:

- Modify the example and configure as end devices one or both modules connected to the button (ROUTER_1, ROUTER_2). They can sleep for three seconds and then wake for one second and send the button status.
- Instead of using the button on the board, connect a digital Grove sensor (for example, a motion sensor) to the board.
- Form a larger sensor network by adding more XBee modules and configuring them to send digital data to COORD.

Lab: receive analog data

This section demonstrates how to create an XBee sensor network. Since the kit does not contain a real sensor, you will simulate an analog sensor with the potentiometer of the XBee Grove Development Board.

Note If you have a Grove sensor, you can connect it to the board for a more realistic example of a sensor network.

For this example, configure one module as coordinator, another as router, and another as end device. The router and end device, where the sensor is connected, will be the senders. The end device will sleep for five seconds, and then wake for the minimum time required to transmit the value of the potentiometer to the coordinator (the receiver).

If you get stuck, see Troubleshooting.

Step 1: Requirements

For this setup you need the following hardware and software.

Hardware

- Three XBee Zigbee Mesh Kit modules
- Three XBee Grove Development Boards
- Three micro USB cables
- One computer

Software

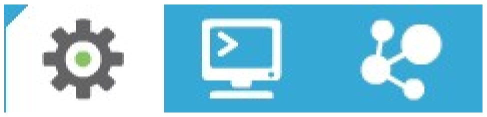
- XCTU 6.3.1 or later
- XBee Java Library (XBJL-X.Y.Z.zip release file)
- Java Virtual Machine 6 or later
- A Java IDE (such as Eclipse or NetBeans)

Tip For more information about XCTU, see the XCTU walkthrough.

Step 2: Connect the components

If you have a Grove sensor (not included in the kit), plug it into the Grove AD2 connector of the XBee B or XBee C (senders) board. Follow the steps to connect your modules:

1. Plug the XBee modules into the XBee Grove Development Boards and connect them to your computer using the micro USB cables provided. You can find more specific steps in Plug in the XBee module.
2. After connecting the modules to your computer, open XCTU.
3. Make sure you are in **Configuration working mode**.



Step 3: Configure the XBee modules

XBee B sends the analog value read from the potentiometer to XBee A, the coordinator, every five seconds. XBee C, the end device, sleeps for five seconds. After this sleep period, it sends the analog value read from the potentiometer to XBee A, then immediately enters low-power mode for another five seconds.

Set the destination address (**DH** + **DL**) of the senders (XBee B and XBee C) to the MAC address (**SH** + **SL**) of the receiver (XBee A). Additionally, configure the pin where the potentiometer connects (DIO3/AD3) as an analog input, and set the sample rate parameter (**IR**) to five seconds.

1. Restore the default settings of all XBee modules with the **Load default firmware settings** button at the top of the Radio Configuration section.

2. Use XCTU to configure the following parameters:

Param	XBee A	XBee B	XBee C	Effect
ID	2015	2015	2015	Defines the network that a radio will attach to. This must be the same for all radios on your network.
JV	—	Enabled [1]	Enabled [1]	Verifies if a coordinator exists on the same channel to join the network or to leave if it cannot be found.
CE	Enabled [1]	—	—	Sets the device as coordinator.
DH	—	0	0	Defines the destination address (high part) to transmit the data to. Use the address 0000000000000000 to address the coordinator.
DL	—	0	0	Defines the destination address (low part) to transmit the data to. Use the address 0000000000000000 to address the coordinator.
NI	COORD	ROUTER	END_ DEVICE	Defines the node identifier, a human-friendly name for the module. ⚠ The default NI value is a blank space. Make sure to delete the space. when you change the value.
AP	API enabled [1]	API enabled [1]	API enabled [1]	Enables API operating mode.
SP	1F4	1F4	1F4	Defines the duration of time spent sleeping. 1F4 (hexadecimal) = 500 (decimal) x 10 ms = 5 seconds.
SM	—	—	Cyclic sleep [4]	Enables the cyclic sleep mode.

Param	XBee A	XBee B	XBee C	Effect
ST	—	—	A	Defines the period of inactivity (no serial or RF data received) before going to sleep. A (hexadecimal) = 10 (decimal) x 1 ms = 10 milliseconds.
D2/D3	—	ADC [2]	ADC [2]	Sets the DIO2/AD2 or DIO3/AD3 pin as ADC in XBee B and XBee C, depending on if the XBee modules are THT or SMT. This pin is connected to a potentiometer. ⚠ Configure the **D2** parameter as ADC [2] only if the XBee module is surface-mount (SMT). However, if the module is through-hole (THT), you have to configure the **D3** parameter as ADC [2] instead of the **D2**.
IR	—	1388	1388	Configures XBee B and XBee C to send IO samples every five seconds (5000 ms = 1388 in hexadecimal).

Note The dash (**—**) in the table means to keep the default value. Do not change the default value.

Note If you are using a Grove sensor and have connected it to the Grove AD2 connector, you must configure the **D2** parameter as ADC [2] instead of the **D3**.

3. Write the settings of all XBee modules with the **Write radio settings** button at the top of the Radio Configuration section.

Step 4: Create a Java project

Create an empty Java project named using Eclipse or NetBeans, with the following project name: **ReceiveAnalogData**.

Option 1: Eclipse

a. Select **File > New**, and click the **Java Project**.
b. The **New Java Project** window appears. Enter the Project name.
c. Click **Next**.

or

Option 2: NetBeans

a. Select **File > New project....**
b. The New Project window appears. In the Categories frame, select **Java > Java Application** from the panel on the right, and click **Next**.
c. Enter the Project name and the Project Location. Clear the Create Main Class option; you will create this later.
d. Click **Finish** to create the project. The window closes and the project appears in the Projects view list on the left side of the IDE.

Step 5: Link libraries to the project

This topic describes how to link the **XBee Java Library**, the **RXTX library** (including the native one), and the **logger library** to the project.

1. Download the XBJL_X.Y.Z.zip library.
2. Unzip the XBJL_X.Y.Z.zip library.
3. Link the libraries using Eclipse or NetBeans:

Option 1: Eclipse

a. Go to the **Libraries** tab of the New Java Project window.
b. Click **Add External JARs....**
c. In the JAR Selection window, search the folder where you unzipped the XBee Java Library and open the **xbee-java-library-X.Y.Z.jar** file.
d. Click **Add External JARs...** again.
e. Go to the **extra-libs** folder and select the following files:
   - **rxtx-2.2.jar**
   - **slf4j-api-x.y.z.jar**
   - **slf4j-nop-x.y.z.jar**
f. Expand the **rxtx-2.2.jar** file of the Libraries tab list, select **Native library location**, and click **Edit....**
g. Click **External folder...** to navigate to the **extra-libs\native\Windows\win32** folder of the directory where you unzipped the XBee Java Library file (XBJL_X.Y.Z.zip).
h. Replace **Windows\win32** with the directory that matches your operating system and the Java Virtual Machine installed (32 or 64 bits). If you don't know which Java Virtual Machine is

installed in your computer, open a terminal or command prompt and execute:

```
java -version
```

i. Click **OK** to add the path to the native libraries.
j. Click **Finish**.

or

Option 2: NetBeans

a. From **Projects** view, right-click your project and go to **Properties**.
b. In the categories list on the left, go to **Libraries** and click **Add JAR/Folder**.
c. In the **Add JAR/Folder** window, search the folder where you unzipped the XBee Java Library and open the **xbjlib-X.Y.X.ja**r file.
d. Click **Add JAR/Folder** again.
e. Go to the **extra-libs** folder and select the following files:
   - **rxtx-2.2.jar**
   - **slf4j-api- x.y.z .jar**
   - **slf4j-nop- x.y.z .jar**
f. Select **Run** in the left tree of the **Properties** dialog.
g. In the **VM Options** field, add the following option:

```
-Djava.library.path=<path_where_the_XBee_Java_Library_is_unzipped>\extra-
libs\native\Windows\win32
```

where:
   - **<path_where_the_XBee_Java_Library_is_unzipped>** is the absolute path of the directory where you unzipped the XBee Java Library file (XBJL_X.Y.Z.zip)
   - **Windows\win32** is the directory that matches your operating system and the Java Virtual Machine installed (32 or 64 bits). If you don't know which Java Virtual Machine is installed in your computer, open a terminal or command prompt and execute:

   ```
   java -version
   ```

h. Click **OK**.

Step 6: Add the source code to the project

Follow these steps to add the source to the project.

1. Open the following source code, select all, and copy it to the clipboard: MainApp.java.
2. Add the Java source file with Eclipse or NetBeans.

Option 1: Eclipse

a. In the **Package Explorer** view, select the project and right-click.
b. From the context menu, select **New > Class**. The **New Java Class** wizard opens.
c. Type the **Name** of the class: **MainApp**.
d. Click **Finish**.

e. The **MainApp.java** file is automatically opened in the editor. Replace its contents with the source code you copied in the previous step.
f. A line at the top of the pasted code is underlined in red. Click on that line; a pop-up appears. Select the first option (**Move 'MainApp.java' to package '...'**) to resolve the error.

Option 2: NetBeans

a. In the **Projects** view, select the project and right-click.
b. From the context menu, select **New > Java Class...** The New Java Class wizard opens.
c. Modify the **Class Name** to be **MainApp**.
d. Click **Finish**.
e. The **MainApp.java** file automatically opens in the editor. Replace its contents with the source code you copied in the previous step.
f. A line at the top of the pasted code is underlined in red. Click on the light bulb next to that line; a pop-up appears. Select the first option (**Move class to correct folder**) to resolve the error.

Step 7: Set the port name and launch the application

For this step, set the port name and launch the application.

1. Change the port name in the Java source code to match the port that the receiver (COORD) is connected to.

```
// TODO: Replace with the port where your coordinator module is connected
private static final String PORT = "COM1";
// TODO: Replace with the baud rate of your coordinator module.
private static final int BAUD_RATE = 9600
```

Note If you are using a Grove sensor connected to AD2, make this additional code change to select the correct IO Line:

2. Launch the application on your computer. Every five seconds, you will receive the value of the potentiometer connected to ROUTER and END_DEVICE (senders). Rotate them and see that the values change.
3. The output of the application should be similar to the following:
4.

```
+----------------------------------------+
|    Receive Analog Data Sample          |
+----------------------------------------+

WARNING: RXTX Version mismatch
       Jar version = RXTX-2.2pre1
       native lib Version = RXTX-2.2pre2

Analog data from '0013A20011111111': 227
Analog data from '0013A20022222222': 0
Analog data from '0013A20011111111': 113
Analog data from '0013A20022222222': 1023
```

Step 8: Section summary of receiving analog data

In this section, you have learned that:

- All XBee modules have a set of pins that you can use to connect and configure sensors or actuators.
- A sensor is a device that provides a corresponding output as a response to events or changes in quantities. There are two types:
  - Digital sensors return discrete values such as on/off.
  - Analog sensors can return a wide variety of values such as the temperature of a room.
- If, as in this example, you want to read data from an analog sensor, you must configure the selected IO as Analog to Digital Converter (ADC).
- You can obtain analog data from a sensor by configuring the remote XBee module to transmit the IO data:
  - To the local device, by setting the **DH** and **DL** parameters to the MAC of the receiver module.
  - Periodically, as you did in this example (using the **IR** or IO Sampling Rate parameter).

Note Remember that the Digital IO Change Detection (**IC**) feature only works for digital pins, so in this case you would not receive any data.

- In this case, the data sent from one module to the other is called IO Sample. It contains the inputs (DIO lines or ADC channels) for which sampling has been enabled. It also contains the value of all enabled digital and analog inputs.
- A sleeping end device will transmit periodic IO samples at the **IR** rate until the Time Before Sleep (**ST**) timer expires and the device can resume sleeping.

Step 9: Do more with receiving analog data

If you are ready to work more extensively with receiving analog data, try the following:

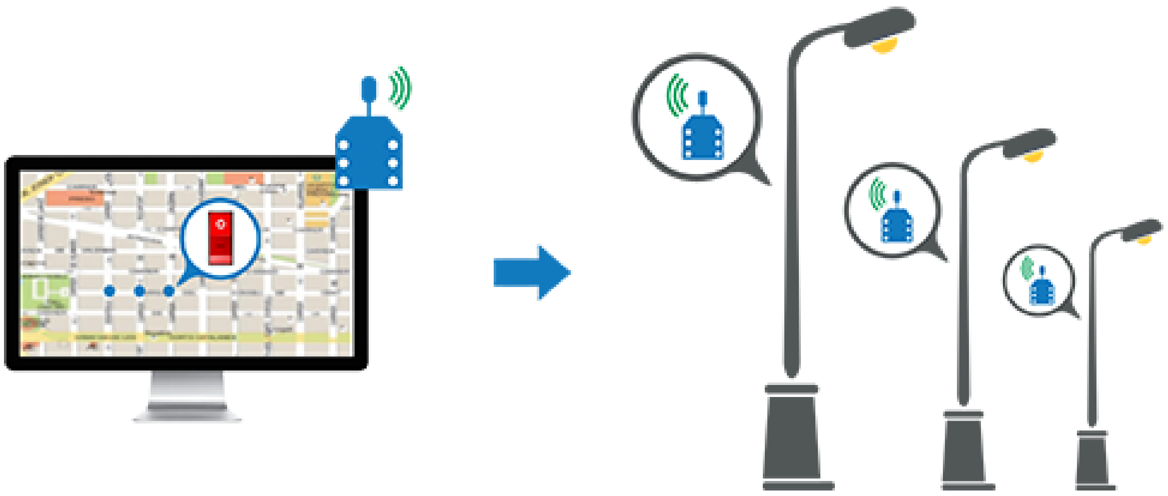
- Instead of using the potentiometer on the board, connect an analog Grove sensor to the board to create a home automation system. You can monitor a number of factors, such as:
  - Temperature
  - Humidity
  - Luminance
  - CO2
  - UV
  - Gas

  You can find more analog sensors at SeeedStudio.
- Form a larger sensor network by adding more XBee modules and configuring them to send analog data to COORD (XBee A).

How XBee modules control devices

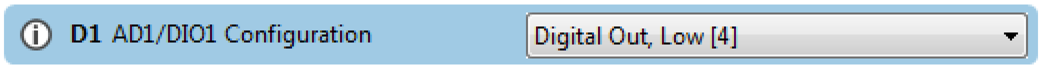
There are many reasons to create a sensor network—that is, to collect data from multiple nodes and bring it to a central location. There are also many reasons you may want to take remote commands from a central location and create real events in multiple physical locations.

An XBee module is capable of receiving commands that set its digital and analog output pins to trigger real-world events without the use of an external microcontroller. By itself, an XBee device can power an LED, sound a small buzzer, or even operate a tiny motor. If you use a relay, you can operate many more devices directly from the XBee module.

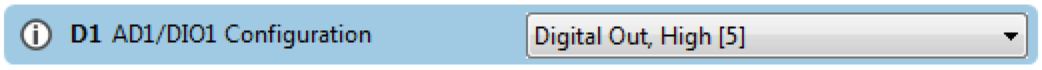


Configure a pin for digital output

If you want to control a digital device, for example to switch a street lamp on or off, connect the device to a pin that supports digital output. XBee Zigbee devices have 15 digital outputs (from **D0** to **D9** and **P0** to **P4**). In addition, configure that pin as Digital Output Low (Digital Out, Low [4]) or Digital Output High (Digital Out, High [5]), depending on the desired default state.



D1 AD1/DIO1 Configuration Digital Out, Low [4]



D1 AD1/DIO1 Configuration Digital Out, High [5]

How to send actuations

Once you configure the pin of a remote device, you can send any actuation to that XBee device. You can use XCTU or the XBee Java Library.

To send an actuation in XCTU, create a Remote AT Command frame in the API console, configuring the following parameters:

- 64-bit destination address: the remote module's MAC address.
- AT command (ASCII): the AT command corresponding to the IO line you want to change (**D0-D9** or **P0-P4**). For example, if you want to send an actuation to DIO3, type **D3**.
- Parameter value (HEX): the new value for the selected IO line is **04** for off, **05** for on.

Example: send digital actuations

In this example, you will create a Java application that blink the LED of a remote XBee module. Configure one of the modules as a coordinator and the other as router. The coordinator is the sender and transmits the digital actuation to the router, which is connected to the LED.

If you get stuck, see Troubleshooting.

Step 1: Requirements

For this setup you need the following hardware and software.

Hardware

- Two XBee Zigbee Mesh Kit modules
- Two XBee Grove Development Boards
- Two micro USB cables
- One computer

Software

- XCTU 6.3.1 or later
- XBee Java Library (XBJL-X.Y.Z.zip release file)
- Java Virtual Machine 6 or later
- A Java IDE (such as Eclipse or NetBeans)

Tip For more information about XCTU, see the XCTU walkthrough.

Step 2: Connect the components

To get started, connect the components and start XCTU.

1. Plug the XBee modules into the XBee Grove Development Boards and connect them to your computer using the micro USB cables provided. You can find more specific steps in Plug in the XBee module.
2. After connecting the modules to your computer, open XCTU.
3. Make sure you are in **Configuration working mode**.

Step 3: Configure the XBee modules

Follow these steps to configure XBee A, the coordinator, to send digital actuations to XBee B, the router, every second. The LED of the receiver dims with each signal received.

1. Restore the default settings of all XBee modules with the **Load default firmware settings** button at the top of the Radio Configuration section.
2. Use XCTU to configure the following parameters:

Param	XBee A	XBee B	Effect
ID	2015	2015	Defines the network that a radio will attach to. This must be the same for all radios in your network.
JV	—	Enabled [1]	Verifies if a coordinator exists on the same channel to join the network or to leave if it cannot be found.
CE	Enabled [1]	—	Sets the device as coordinator.
NI	COORD	ROUTER	Defines the node identifier, a human-friendly name for the module. ⚠ The default **NI** value is a blank space. Make sure to delete the space when you change the value.
AP	API Enabled [1]	API Enabled [1]	Enables API mode.
D4	—	Digital Out, High [5]	Sets the DIO4/AD4 pin as Digial Output High in XBee B. The LED is connected to this pin.

Note The dash (**—**) in the table means to keep the default value. Do not change the default value.

3. Write the settings of all XBee modules with the **Write radio settings** button at the top of the Radio Configuration section.

Step 4: Create a Java project

Option 1: Eclipse

a. Select **File > New**, and click the **Java Project**.
b. The **New Java Project** window appears. Enter the Project name.
c. Click **Next**.

or

Option 2: NetBeans

a. Select **File > New project....**
b. The New Project window appears. In the Categories frame, select **Java > Java Application** from the panel on the right, and click **Next**.

c. Enter the Project name and the Project Location. Clear the Create Main Class option; you will create this later.
d. Click **Finish** to create the project. The window closes and the project appears in the Projects view list on the left side of the IDE.

Step 5: Link libraries to the project

This topic describes how to link the **XBee Java Library**, the **RXTX library** (including the native one), and the **logger library** to the project.

1. Download the XBJL_X.Y.Z.zip library.
2. Unzip the XBJL_X.Y.Z.zip library.
3. Link the libraries using Eclipse or NetBeans:

Option 1: Eclipse

a. Go to the **Libraries** tab of the New Java Project window.
b. Click **Add External JARs....**
c. In the JAR Selection window, search the folder where you unzipped the XBee Java Library and open the **xbee-java-library-X.Y.Z.jar** file.
d. Click **Add External JARs...** again.
e. Go to the **extra-libs** folder and select the following files:
   - **rxtx-2.2.jar**
   - **slf4j-api-x.y.z.jar**
   - **slf4j-nop-x.y.z.jar**
f. Expand the **rxtx-2.2.jar** file of the Libraries tab list, select **Native library location**, and click **Edit....**
g. Click **External folder...** to navigate to the **extra-libs\native\Windows\win32** folder of the directory where you unzipped the XBee Java Library file (XBJL_X.Y.Z.zip).
   - Replace **Windows\win32** with the directory that matches your operating system and the Java Virtual Machine installed (32 or 64 bits). If you don't know which Java Virtual Machine is installed in your computer, open a terminal or command prompt and execute:

```
java -version
```

h. Click **OK** to add the path to the native libraries.
i. Click **Finish**.

or

Option 2: NetBeans

a. From **Projects** view, right-click your project and go to **Properties**.
b. In the categories list on the left, go to **Libraries** and click **Add JAR/Folder**.
c. In the **Add JAR/Folder** window, search the folder where you unzipped the XBee Java Library and open the **xbjlib-X.Y.X.ja**r file.
d. Click **Add JAR/Folder** again.

e. Go to the **extra-libs** folder and select the following files:
   - **rxtx-2.2.jar**
   - **slf4j-api- x.y.z .jar**
   - **slf4j-nop- x.y.z .jar**

f. Select **Run** in the left tree of the **Properties** dialog.

g. In the **VM Options** field, add the following option:

```
-Djava.library.path=<path_where_the_XBee_Java_Library_is_unzipped>\extra-
libs\native\Windows\win32
```

where:
   - **<path_where_the_XBee_Java_Library_is_unzipped>** is the absolute path of the directory where you unzipped the XBee Java Library file (XBJL_X.Y.Z.zip)
   - **Windows\win32** is the directory that matches your operating system and the Java Virtual Machine installed (32 or 64 bits). If you don't know which Java Virtual Machine is installed in your computer, open a terminal or command prompt and execute:

```
java -version
```

h. Click **OK**.

Step 6: Add the source code to the project

Follow these steps to add the source to the project.

1. Open the following source code, select all, and copy it to the clipboard: MainApp.java
2. Add the Java source file with Eclipse or NetBeans.

Option 1: Eclipse

a. In the **Package Explorer** view, select the project and right-click.

b. From the context menu, select **New > Class**. The **New Java Class** wizard opens.

c. Type the **Name** of the class: **MainApp**.

d. Click **Finish**.

e. The **MainApp.java** file is automatically opened in the editor. Replace its contents with the source code you copied in the previous step.

f. A line at the top of the pasted code is underlined in red. Click on that line; a pop-up appears. Select the first option (**Move 'MainApp.java' to package '...'**) to resolve the error.

Option 2: NetBeans

a. In the **Projects** view, select the project and right-click.

b. From the context menu, select **New > Java Class...** The New Java Class wizard opens.

c. Modify the **Class Name** to be **MainApp**.

d. Click **Finish**.

e. The **MainApp.java** file automatically opens in the editor. Replace its contents with the source code you copied in the previous step.

f. A line at the top of the pasted code is underlined in red. Click on the light bulb next to that line; a pop-up appears. Select the first option (**Move class to correct folder**) to resolve the error.

Step 7: Set the port name and launch the application

For this step, set the port name and launch the application.

1. Change the port name in the Java source code to match the port that the COORD module (sender) is connected to.

   ```
   // TODO: Replace with the port where your coordinator module is
   connected.
   private static final String PORT = "COM1";
   // TODO: Replace with the baud rate of your coordinator module.
   private static final int BAUD_RATE = 9600
   ```

2. Launch the application in your computer. Every second, you will see that the LED of the ROUTER module (receiver) changes state.

Step 8: Section summary of sending digital actuations

In this section, you have learned that:

- All XBee modules have a set of pins you can use to connect and configure sensors or actuators.
- An actuator is a device that controls a mechanism or system. For instance, you can use an XBee module connected to an actuator to send digital information to another XBee module so that it raises or lowers your window blinds.
- You can create a network with a central node that sends orders to remote nodes. This network allows you to trigger real-world events wirelessly, such as switching on all the lights at home, via actuators connected to remote node(s).
- The default state of a pin that supports digital output depends on the values of the DIO setting:
  - Digital Output, Low [4]: the output is set by default to low.
  - Digital Output, High [5]: the output is set by default to high.

Step 9: Do more with sending digital actuations

If you are ready to work more extensively with actuations, try the following:

- Add sensors to your network, as explained in Example: receive digital data and Lab: receive analog data. Then control your actuators depending on the value returned. For example, switch the router's LED on when a button connected to coordinator is pressed, or when the value of the coordinator's potentiometer exceeds a defined threshold.
- Instead of controlling an LED, connect a relay to one of the digital output pins to create a home automation system. You can:
  - Switch lights on/off.
  - Switch the irrigation system of your garden on/off.
  - Raise/lower the blinds.
  - Control your garage door.
- Extend the network by adding more XBee devices connected to different devices.

- Control all your devices remotely with a smartphone application connected to an XBee Gateway. See Related products.

Security and encryption

Zigbee supports various levels of security that you can configure depending on the needs of the application. Security provisions include:

- 128-bit AES encryption
- Two security keys that can be preconfigured or obtained during joining
- Trust center support
- Provisions to ensure message integrity, confidentiality, and authentication

Zigbee security model

The Zigbee standard supports three security modes:

- **Residential security** was first supported in the Zigbee 2006 standard. This level of security requires that a network key be shared among devices.
- **Standard security** adds a number of optional security enhancements over residential security, including an APS layer link key.
- **High security** adds entity authentication, and a number of other features not widely supported.

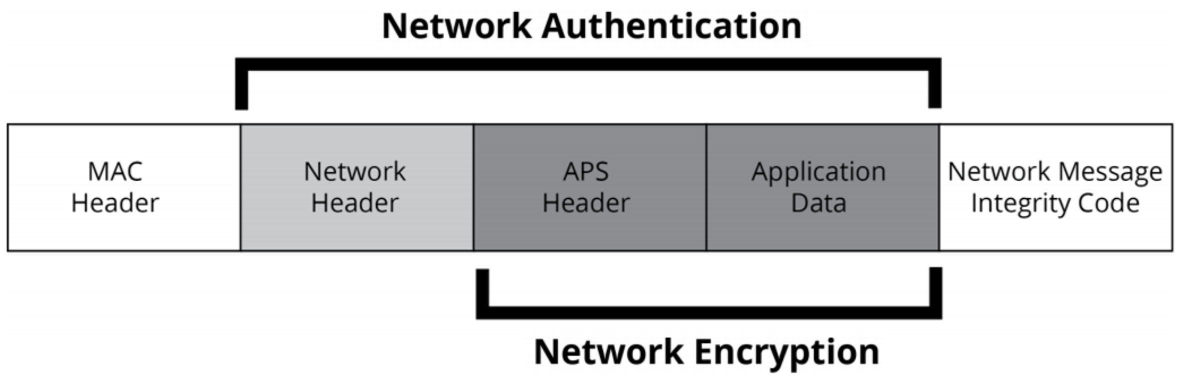
XBee Zigbee modules primarily support standard security, so we focus on this mode.

Note If you want to learn more about Zigbee security, see *Chapter 4 - Security Services Specification* of the Zigbee specification.

Zigbee security is applied to the Network and APS layers, and packets are encrypted with 128-bit AES encryption. A network key and optional link key can be used to encrypt data. Only devices with the same keys are able to communicate together in a network.

Network layer security

The XBee device uses the network key to encrypt the APS layer and application data. If security is enabled in a network, all data packets will be encrypted with the network key using 128-bit AES.

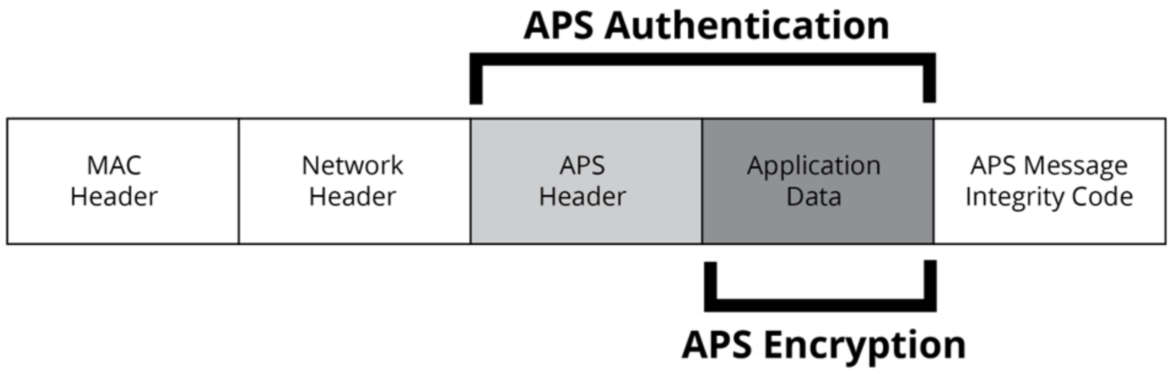


The network header, APS header, and application data are all authenticated with 128-bit AES. A hash is performed on these fields and is appended as a four-byte message integrity code (MIC) to the end of the packet. The MIC allows receiving devices to ensure the message has not been changed.

Packets with network layer encryption are encrypted and decrypted by each hop in a route. When a device receives a packet with network encryption, it decrypts the packet and authenticates the packet. If the device is not the destination, it then encrypts and authenticates the packet. Since network encryption is performed at each hop, packet latency is slightly longer in an encrypted network than in a non-encrypted network.

APS layer security

APS layer security can be used to encrypt application data using a key that is shared between source and destination devices. Where network layer security is applied to all data transmissions and is decrypted and reencrypted on a hop-by-hop basis, APS security is optional and provides end-to-end security using an APS link key known by only the source and destination device. APS security cannot be applied to broadcast transmissions.



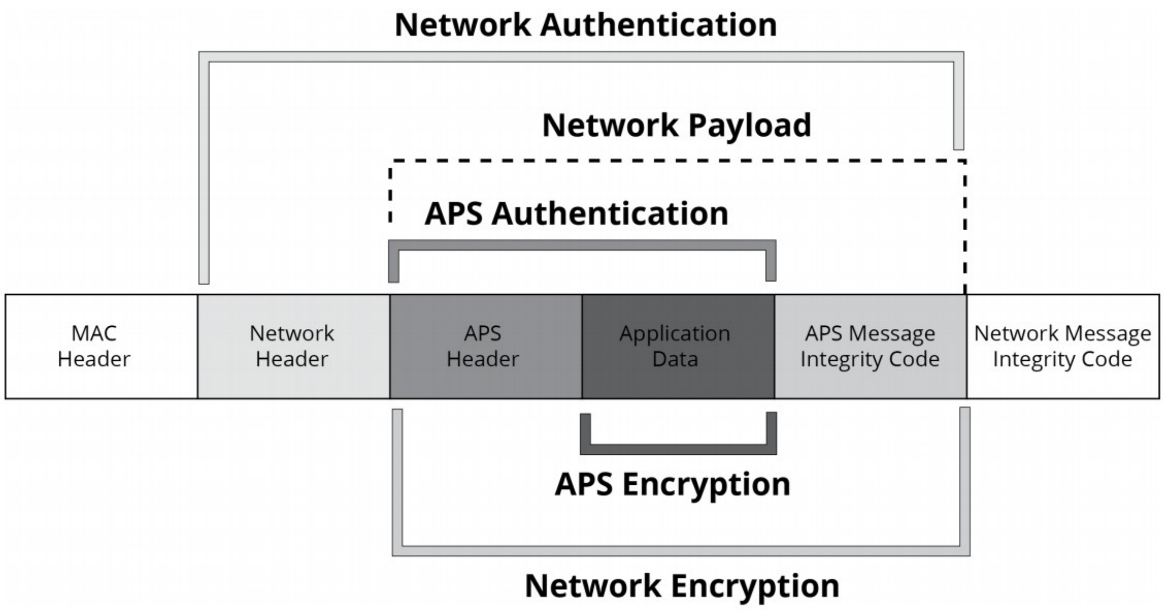
If APS security is enabled, the APS header and data payload are authenticated with 128-bit AES. A hash is performed on these fields and appended as a four-byte message integrity code (MIC) to the end of the packet. This MIC is different than the MIC appended by the network layer. The MIC allows the destination device to ensure the message has not been changed.

There are two kinds of APS link keys – trust center link keys and application link keys. A trust center link key is established between a device and the trust center, and an application link key is established between a device and another device in the network where neither device is the trust center.

Note Zigbee defines a trust center device that is responsible for authenticating devices that join the network. The trust center also manages link key distribution in the network.

Network and APS layer encryption

Network and APS layer encryption can both be applied to data. The following figure demonstrates the authentication and encryption performed on the final Zigbee packet when both are applied.



Form or join a secure network

The coordinator is responsible for selecting a network encryption key. This key can either be preconfigured or randomly selected. In addition, the coordinator generally operates as a trust center and must select the trust center link key. The trust center link key can also be preconfigured or randomly selected.

Devices that join the network must obtain the network key when they join. When a device joins a secure network, the network and link keys can be sent to the joining device. If the joining device has a preconfigured trust center link key, the network key will be sent to the joining device encrypted by the link key. Otherwise, if the joining device is not preconfigured with the link key, the device could only join the network if the network key is sent unencrypted—"in the clear".

The trust center must decide whether or not to send the network key unencrypted to joining devices that are not preconfigured with the link key. We do not recommend sending the network key unencrypted as it can open a security hole in the network. To maximize security, preconfigure devices with the correct link key.

Security on the XBee

If you enable security in the XBee Zigbee firmware, devices acquire the network key when they join a network. Data transmissions are always encrypted with the network key, and can optionally be end-to-end encrypted with the APS link key.

Enable security

To enable security on a device, the Encryption Enable (**EE**) parameter must be set to 1. When the parameter value changes, the XBee module leaves the network (PAN ID and channel) it was operating on and attempt to form or join a new network. If you set **EE** to 1, all data transmissions are encrypted with the network key.

Note The **EE** parameter must be set the same on all devices in a network.

Set the network security key

The coordinator selects the network security key for the network using the Network Encryption Key (**NK**) parameter (write-only). If NK=0 (default), the coordinator will selects a random network key. Otherwise, you set **NK** to a non-zero value, it uses this value as network security key.

NK is only supported on the coordinator. Routers and end devices with security enabled (EE=1) acquire the network key when they join a network. They receive the network key encrypted with the link key if they share a preconfigured link key with the coordinator.

Set the APS trust center link key

The coordinator must also select the trust center link key, using the Encryption Key (**KY**) parameter (write-only). If KY=0 (default), the coordinator select a random trust center link key (not recommended). Otherwise, if you set **KY** greater than 0, the module uses this value as the preconfigured trust center link key.

If the coordinator selects a random trust center link key (KY=0, default), then it allows devices to join the network without having a preconfigured link key. However, sends the network key unencrypted over-the-air to joining devices and is not recommended.

If the coordinator uses a preconfigured link key (KY > 0), then the coordinator will not send the network key unencrypted to joining devices. Only devices with the correct preconfigured link key can able to join and communicate on the network.

Enable APS encryption

APS encryption is an optional layer of security that uses the link key to encrypt the data payload. Unlike network encryption that is decrypted and encrypted on a hop-by-hop basis, APS encryption is only decrypted by the destination device. The XBee must be configured with security enabled (**EE** set to 1) to use APS encryption.

APS encryption can be enabled in API firmware on a per-packet basis. To enable APS encryption for a given transmission, set the "enable APS encryption" transmit options bit in the API transmit frame. Enabling APS encryption decreases the maximum payload size by nine bytes.

Use a trust center

Use the Encryption Options (**EO**) parameter define the coordinator as a trust center. If the coordinator is a trust center, it received alerts to all new join attempts in the network. The trust center also has the ability to update or change the network key on the network.

How to update the network key with a trust center.

If the trust center has started a network and the **NK** value changes, the coordinator updates the network key on all devices in the network. Changes to **NK** will not force the device to leave the network. The network continues to operate on the same channel and PAN ID, but the devices in the network update their network key, increment their network key sequence number, and restore their frame counters to 0.

How to update the network key without a trust center.

If the coordinator is not running as a trust center, the Network Reset (**NR1**) command can be used to force all devices in the network to leave the current network and rejoin the network on another channel. When devices leave and reform then network, the frame counters are reset to 0. This approach causes the coordinator to form a new network that the remaining devices should join. Resetting the network in this manner brings the coordinator and routers in the network down for about ten seconds, and causes the 16-bit PAN ID and 16-bit addresses of the devices to change.

In Zigbee firmware, a secure network can be established with or without a trust center. Network and APS layer encryption are supported regardless of whether a trust center is used.

Example: basic (but secure) communication

In this example, add security to the Example: basic communication by encrypting communication between the three XBee modules. Note that this feature is applicable for both **AT** and API operating modes.

First, follow the steps explained in the Basic Communication example. When you have added the modules to XCTU and changed the value of the corresponding settings, enable security on each module. To do so, set the **EE**, **KY**, and **NK** parameters as follows and write the new values:

Param	XBee A	XBee B	XBee C	Effect
EE	Enabled [1]	Enabled [1]	Enabled [1]	Enables Zigbee encryption.
KY	1234567890	1234567890	1234567890	Sets the APS trust center link key. You can use any 32 hexadecimal string.
NK	0	—	—	Sets the network security key. In this case, the coordinator selects a random key and sends it to the other modules encrypted with the preconfigured link key when they join the network.

Note The dash (**—**) in the table means to keep the default value. Do not change the default value.

Once you have done this, all communications between the XBee modules are encrypted with ther andom network key selected by the coordinator.

Understanding the example

Once you have completed the steps outlined in the Basic (but secure) communication example, send a message from ROUTER or END_DEVICE to COORD. You will see that the message was received correctly but you will not detect the encryption/decryption process or notice a difference in the way the XBees display the information. Now try disabling the encryption on ROUTER or END_DEVICE. Or, change the Encryption Key (KY) on one of them. In both of these instances, the receiver module (COORD) will not receive the messages.

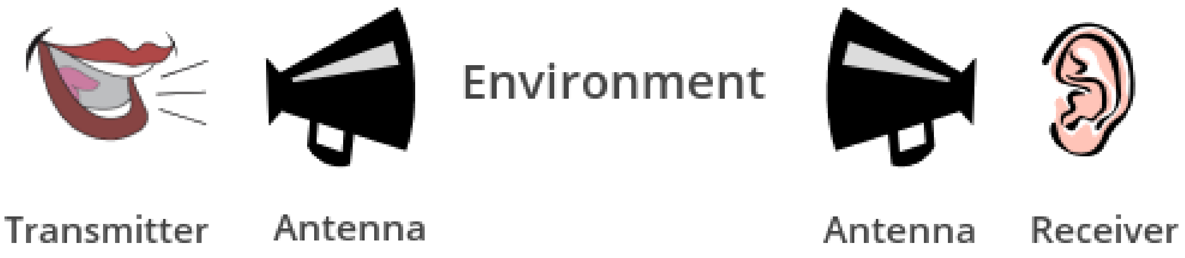
Signal strength and radio frequency range

This section describes how obstacles and other factors can impact how well the devices in your network communicate. Once you learn about the factors that can impact your signal and wireless communications, you can try performing a range test.

Distance and obstacles

Basic communication systems involve the following components:

- Transmitting element
- Receiving device
- Environment through which communication is occurring
- Antennas or other focusing elements



RF communication can be compared to simple audio communication: our vocal cords transmit sound waves that may be received by someone's eardrum. We can use a megaphone to focus and direct the sound waves in order to make the communication more efficient.

The **transmitter**'s role in wireless communication is to feed a signal to an antenna for transmission. A radio transmitter encodes data in RF waves with a certain signal strength (power output) to project the signal to a receiver.

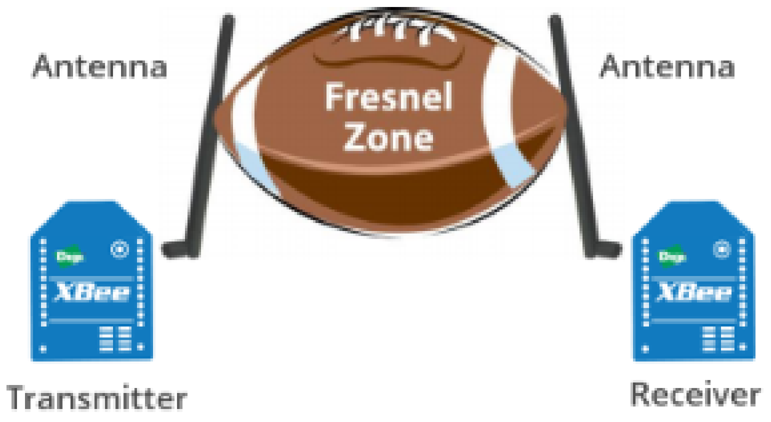
The **receiver** gets and decodes data that comes through the receiving antenna. The receiver performs the task of accepting and decoding designated RF signals while rejecting unwanted ones.

Antennas are devices that focus energy in a particular direction, similar to the way the megaphone focuses voice energy. Antennas can provide different radiation patterns depending on the design and application. How much the energy is focused in a given direction is referred to as antenna gain.

The space between the transmitter and receiver is the system's **environment**. Attaining RF line-of-sight (LOS) between sending and receiving antennas is essential to achieving long range in wireless communication. There are two types of LOS that are generally used to describe an environment:

- **Visual LOS** is the ability to see from one site to the other. It requires only a straight linear path between two points.
- **RF LOS** requires not only visual LOS, but also a football-shaped path called a **Fresnel zone** that is free of obstacles so data can travel optimally from one point to another. The Fresnel

zone can be thought of as a tunnel between two sites that provides a path for RF signals.

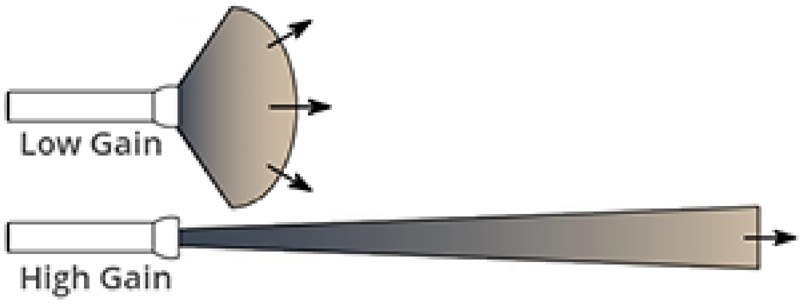


Factors affecting wireless communication

Although the communication distance specified for some XBee devices can be up to 25 miles or more, this value may be affected by factors that may decrease the quality of the signal:

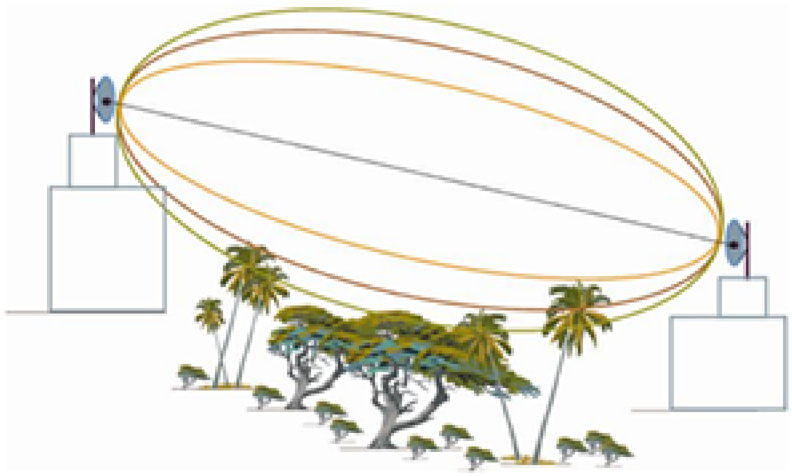
- **Some materials can reflect the radio frequency waves, causing interference** with other waves and loss of signal strength. In particular, metallic or conductive materials are great reflectors, although almost any surface can reflect the waves and interfere with other radio frequency waves.
- **Radio waves can be absorbed by objects in their path, causing loss of power** and limiting transmission distance.
- **Antennas can be adjusted to increase the distance that data can travel in a wireless communication system**. The more focus the antenna can apply, the more range the system will yield. High-gain antennas can achieve greater range than low-gain antennas, although they cover less area.

  A flashlight can help illustrate the principle. Some flashlights allow the user to adjust the beam of light by twisting the lens to focus or spread the beam of light. When the lens spreads—or diffuses—the beam of light, that beam of light travels a shorter distance than when the lens is twisted to focus the beam of light.



- **Line-of-sight can help increase reliability of the signal**.

  To achieve the greatest range, the football-shaped path in which radio waves travel (Fresnel zone) must be free of obstructions. Buildings, trees, or any other obstacles in the path will decrease the communication range. If the antennas are mounted just off the ground, over half of the Fresnel zone ends up being obstructed by the curvature of the earth, resulting in significant reduction in range. To avoid this problem, mount the antennas high enough off the ground that the earth does not interfere with the central diameter of the Fresnel zone.

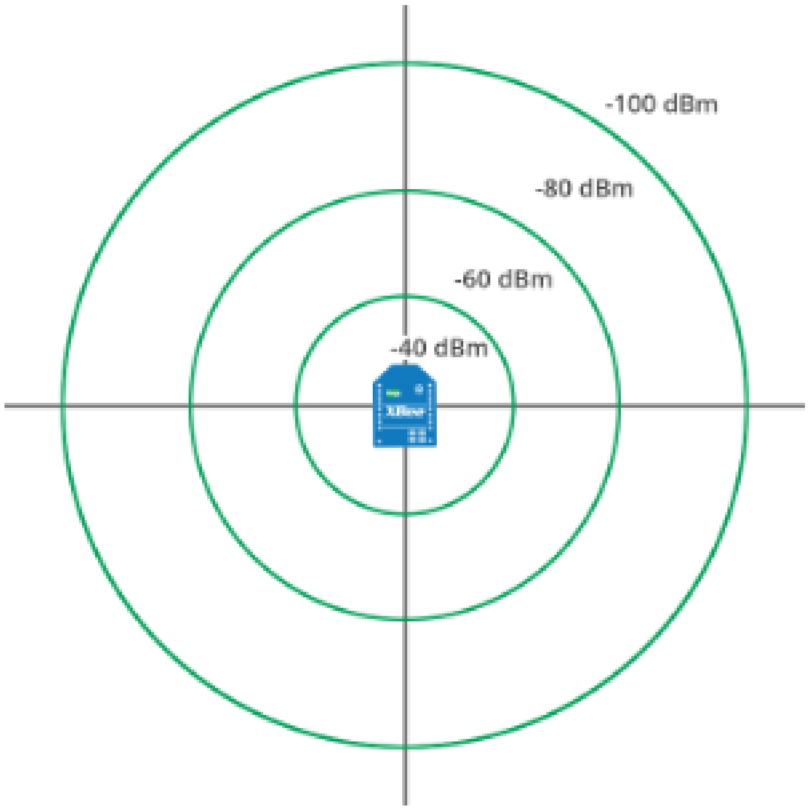


Signal strength and the RSSI pin

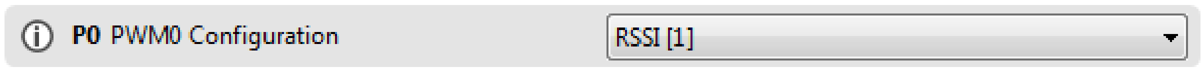
The Received Signal Strength Indicator (RSSI) measures the amount of power present in a radio signal. It is an approximate value for signal strength received on an antenna.

Measuring the signal strength at the receiving antenna is one way to determine the quality of a communication link. If a distant transmitter is moved closer to a receiver, the strength of the transmitted signal at the receiving antenna increases. Likewise, if a transmitter is moved farther away, signal strength at the receiving antenna decreases.

The RSSI is measured in dBm. A greater negative value (in dBm) indicates a weaker signal. Therefore, -50 dBm is better than -60 dBm.

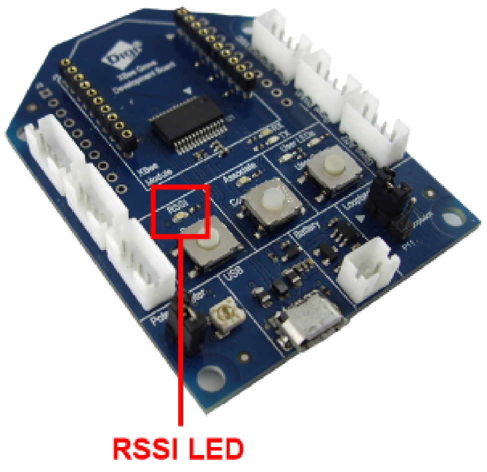


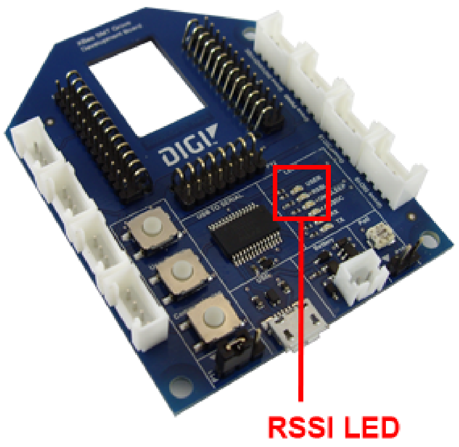
XBee module's pin 6 can be configured as an RSSI pin that outputs a PWM (pulse-width modulation) signal representing this value. To do so, configure **P0** as RSSI [1]:



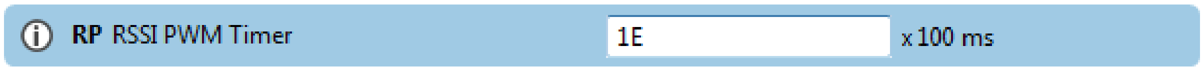
P0 PWM0 Configuration RSSI [1]

The XBee Grove Development Board includes an LED connected to the XBee module's pin 6. When this pin is configured as the RSSI pin, the LED lights every time the connected XBee module receives data. Its intensity represents the RSSI value of the last-received data: a brighter light means a higher RSSI value and better signal quality.





Configure the amount of time the RSSI pin is active, and therefore the amount of time the LED will remain lit, by modifying the RSSI PWM Timer (RP) setting:



RP RSSI PWM Timer 1E x 100 ms

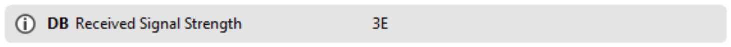
RP value is expressed in hexadecimal notation. For example, a configured value of 0x1E is equivalent to 30 in decimal and means that the pin will be active for three seconds (30*100=3000ms.) So the LED will light for a total of three seconds, representing the last RSSI value.

After the **RP** time has elapsed and no data has been received, the pin will be set to low and the LED will not light until more data is received. The pin will also be set to low at power-up until the first data packet is received. A value of 0xFF permanently enables the pin; when configured in this way, it will always reflect the RSSI value of the last-received data packet.

Although the luminosity variations of the RSSI LED may be difficult to distinguish, the LED can be used to verify successful receipt of data packets. Each time the XBee module receives data, the LED is solid during the configured time.

Note Received Signal Strength (DB) parameter

The RSSI value can also be obtained by reading the XBee **DB** parameter value. It represents the RSSI absolute value of the last received data packet expressed in hexadecimal notation.



DB Received Signal Strength 3E

Is RSSI the best indication of link quality?

One thing to keep in mind is that the RSSI is only an indication of the RF energy detected at the antenna port. The power level reported could be artificially high because it may include energy from background noise and interference as well as the energy from the desired signal. This situation is worse in an interference-prone environment where it is possible to get consistently high RSSI readings, yet still have communication errors.

If the application is attempting to measure "link reliability" and not just "signal strength," it may be helpful to factor in "% packets received" or similar data.

Tip A range test is always a good idea, as it allows you to measure link performance in terms of signal strength and packet success rate. This will help you determine the reliability of your RF system. For more information, see Example: perform a range test.

Range test

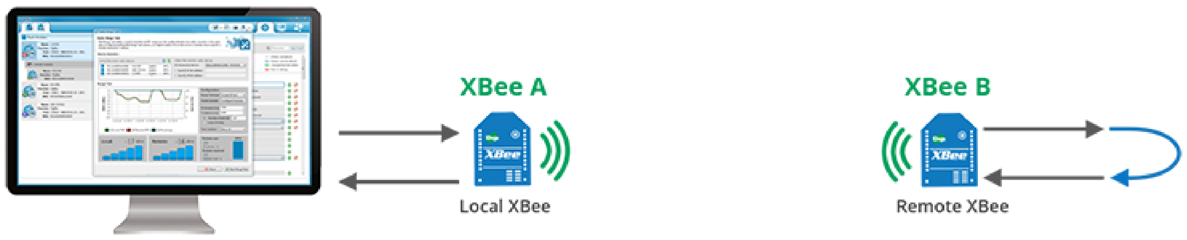
Since the communication between XBee modules takes place over the air, the quality of the wireless signal can be affected by many factors: absorption, reflection of waves, line-of-sight issues, antenna style and location, etc.

A range test demonstrates the real-world RF range and link quality between two XBee modules in the same network. Performing a range test will give an initial indication of the expected communication performance of the kit components.

When deploying an actual network, multiple range tests are recommended to analyze varying conditions in your application.

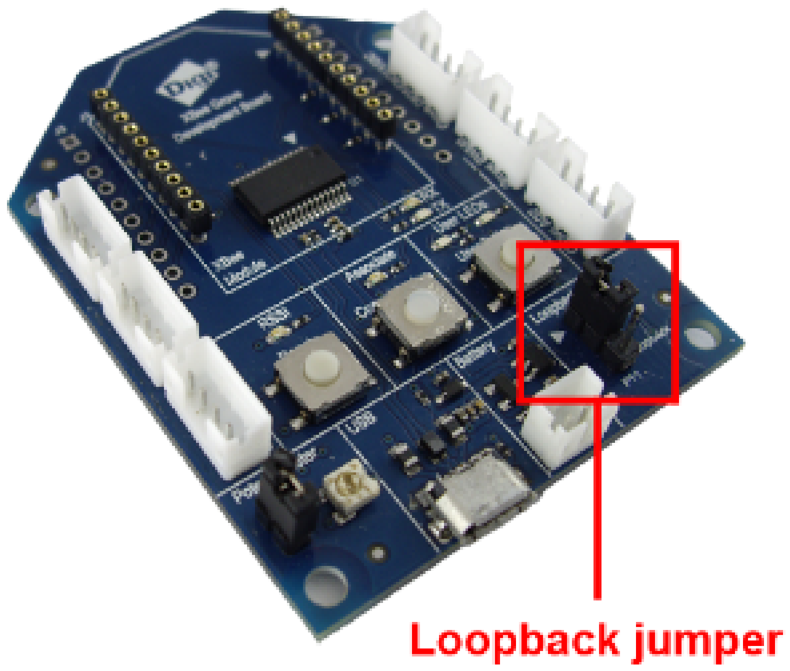
XCTU allows you to perform a range test with at least one XBee module connected to your computer (local) and another remote XBee module, both in the same network. The range test involves sending data packets from the local XBee module to the remote and waiting for the echo to be sent from the remote to the local. During this process, XCTU counts the number of packets sent and received by the local module and measures the signal strength of both sides (RSSI):

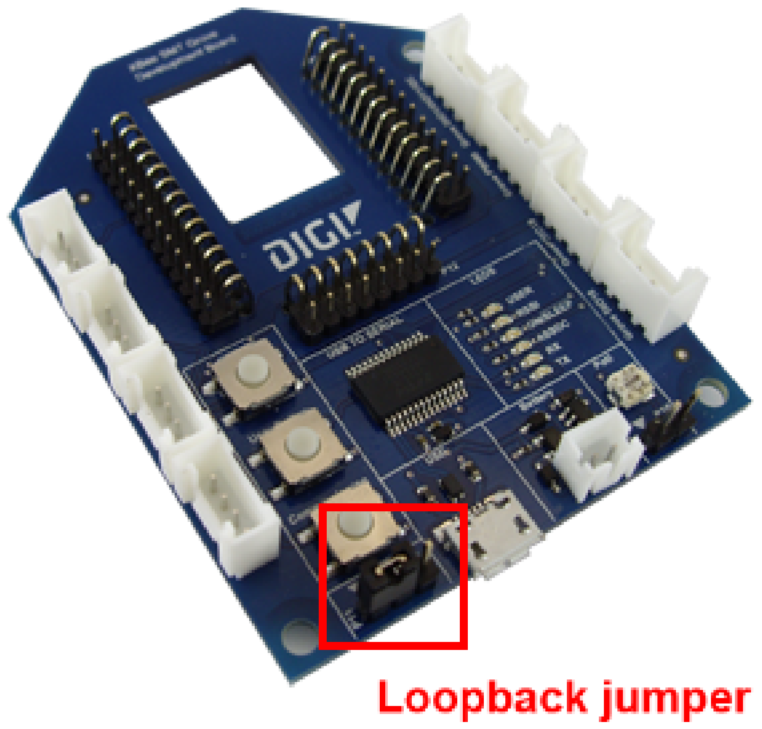
- RSSI is the Received Signal Strength Indicator value.
- Every sent packet from the local XBee module should be received again as an echo by the same local XBee module.



There are two types of range tests:

- **Loopback cluster** (0x12): The range test is performed using explicit addressing frames/packets directed to the Cluster ID 0x12 on the data endpoint (0xE8) which returns the received data to the sender. Not all XBee variants support the Loopback Cluster. XCTU Range Test tool displays an error when this method is selected and the XBee module does not support it.
- **Hardware loopback**: The range test is performed using the serial port/USB hardware loopback capabilities. To use this type, the remote module must be configured to work in transparent mode and the loopback jumper must be closed before starting. This causes any received data to be transmitted back to the sender.

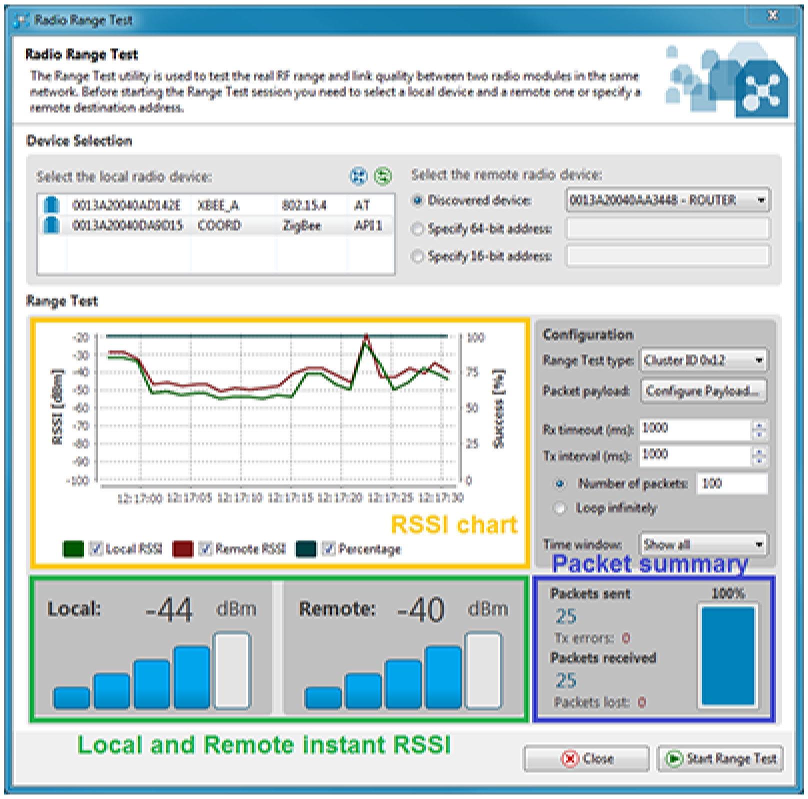




Note The local XBee module (the one attached to your computer) can be configured to use API or transparent mode. The RSSI value of the remote device can only be read when the local XBee module is working in API mode.

Once the range test process has started, XCTU represents the retrieved data in three ways:

- **RSSI Chart** represents the RSSI values of the local and remote devices during the range test session. The chart also contains the percentage of success for the total packets sent.
- **Local and Remote instant RSSI value** display the instant RSSI value of the local and remote devices. This value is retrieved for the last packet sent/received.
- **Packet summary** displays the total number of packets sent, packets received, transmission errors, and packets lost. It also displays the percentage of success sending and receiving packets during the range test session.



Radio Range Test

Radio Range Test

The Range Test utility is used 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:

0013A20040AD142E	XBEE_A	802.15.4	AT
0013A20040DA9D15	COORD	ZigBee	API1

Select the remote radio device:

- [✓ Discovered device:] 0013A20040AA3448 - ROUTER
- Specify 64-bit address:
- Specify 16-bit address:

Range Test

Configuration

Range Test type: Cluster ID 0x12
Packet payload: Configure Payload...
Rx timeout (ms): 1000
Tx interval (ms): 1000
[✓ Number of packets:] 100
Loop infinitely
Time window: Show all

Local: -44 dBm
Remote: -40 dBm

Packets sent 25
Tx errors: 0
Packets received 25
Packets lost: 0
100%

Note For more information about the range test tool, read the XCTU documentation.

Example: perform a range test

Follow the steps in this section to perform a range test with XCTU using the loopback cluster of your XBee modules.

The steps show you how to review the RSSI of both local and remote modules and the number of packets successfully sent and received by the local module during the range test session.

If you get stuck, see Troubleshooting.

Step 1: Requirements

Hardware

- Two XBee Zigbee modules
- Two XBee Grove Development Boards
- Two micro USB cables
- One computer, although you may also use two

Software

- XCTU 6.3.1 or later

Step 2: Connect the components

To get started, connect the components and start XCTU.

1. Plug the XBee modules into the XBee Grove Development Boards and connect them to your computer using the micro USB cables provided. You can find more specific steps in Plug in the XBee module.
2. After connecting the modules to your computer, open XCTU.
3. Make sure you are in **Configuration working mode**.

Step 3: Configure the XBee Zigbee modules

To perform a range test, configure one device as coordinator (XBee A, local) and the other as router (XBee B, remote).

The XBee Zigbee modules support the loopback cluster (0x12). Using this type of range test is advantageous because you do not have to close the loopback jumper of the remote module and the it can work in any operating mode.

To configure your modules to perform the range test, follow these steps:

1. Restore the default settings of all XBee modules with the **Load default firmware settings** button at the top of the Radio Configuration section.
2. Use XCTU to configure the following parameters:

Param	XBee A (local)	XBee B (remote)	Effect
ID	2015	2015	Defines the network a radio will connect to. This parameter must be the same for all radios on your network.

Param	XBee A (local)	XBee B (remote)	Effect
JV	—	Enabled [1]	Verifies if a coordinator exists on the same channel to join the network or to leave if it cannot be found.
CE	Enabled [1]	—	Sets the device as coordinator.
NI	COORD	ROUTER	Defines the node identifier, a human-friendly name for the module. ⚠ The default NI value is a blank space. Make sure to delete the space when you change the value.
AP	API enabled [1]	API enabled [1]	Enables API mode.

3. Write the settings of all XBee modules with the **Write radio settings** button at the top of the Radio Configuration section.

Step 4: Perform a range test

Follow these steps to perform the range test.

1. Open the **Tools** menu within XCTU and select the **Range Test** option.
2. Your local devices are listed on the left side of the Devices Selection section. Select the COORD module and click the **Discover remote devices** button .
3. When the discovery process finishes, the other device (ROUTER) is displayed in the Discovering remote devices... dialog. Click **Add selected devices**.
4. Select the ROUTER module from the Discovered device list located on the right panel inside.
5. Click **Start Range Test**.
6. Range test data is represented in the chart. By default, 100 packets are being sent for the test. The instant local and remote RSSI are also displayed in two separate controls, as well as the number of packets sent and received.
7. Test the signal interference by doing one of the following: place your hands over one of the modules, block line-of-sight with your body, place a metal box over an XBee module, or move the remote XBee module to a further location like a different room or floor of the building. The RSSI value will decrease and some packets may even be lost.
8. Click **Stop Range Test** to stop the process at any time.

Step 5: Section summary of signal strength

In this section, you have learned the following:

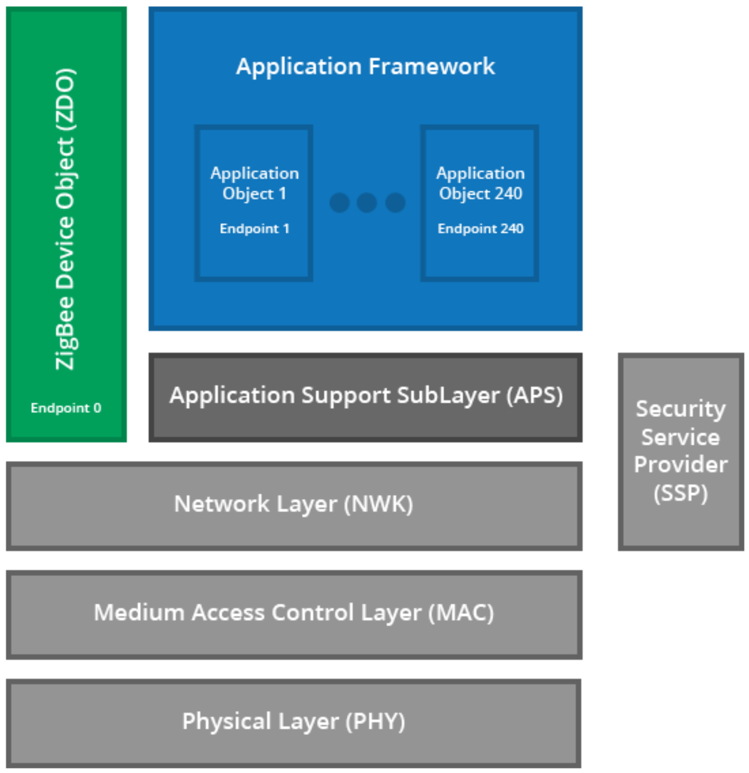
- There are two types of line-of-site (LOS) that describe an environment:
  - Visual LOS describes the ability to see from one place to the other. It requires only a straight linear path.
  - RF LOS requires visual LOS and a Fresnel zone free of obstacles for data to travel optimally.
- Almost any surface, especially metallic or conductive materials, can cause interference and a resulting loss of quality in transmitted data.
- Adjust antennas to increase the distance data can travel. Place them high up off the ground to help increase their range.
- The RSSI or Received Signal Strength Indicator measures the amount of power present in a radio signal. It is measured in dBm, and its depends on the protocol.
- XBee pin 6 (**P0**) can be configured as RSSI. The Grove board includes an LED connected to it which visually represents the RSSI value of the last-received data by varying its light intensity.
- The RSSI PWM Timer (**RP**) parameter helps you configure the amount of time the RSSI pin is active.
- The Received Signal Strength (**DB**) parameter represents the absolute RSSI value, in hexadecimal notation, of the last data packet received.
- A range test determines the real-world RF range and link quality between two XBee modules in the same network by sending data packets from the local device to the remote device and waiting for an echo.
- There are two types of range tests: loopback cluster and hardware loopback. The loopback cluster test is preferable, as it doesn't require you to change the loopback jumper of the remote module's board; however, this type of test is not supported by all XBee variants.

Zigbee communication in depth

Zigbee is a global wireless standard that enables simple and smart objects to work together. This interoperability—multiple devices from different vendors working together to achieve a common goal—is one of the biggest advantages of using the Zigbee protocol.

Imagine that you want to automate your home to control the heating and cooling systems, lights, doors, blinds, irrigation system, etc. For these devices to interoperate, they need to "speak" the same language. Even if the light bulb was manufactured by Company A and the switch was manufactured by Company B, if they are Zigbee-certified products they can work together.

To achieve this interoperability among devices, the Zigbee protocol is organized in layers which separate the components and functions into independent modules. Each layer performs a specific set of services for the layer above.



As mentioned in the first topics of this guide, Zigbee is built on top of the IEEE 802.15.4 standard, so the bottom two layers (Physical Layer (PHY) and Medium Access Control Layer (MAC)) are from that specification. The Network Layer (NWK) from the Zigbee specification, which handles network structure, routing, and security, is just above the MAC layer.

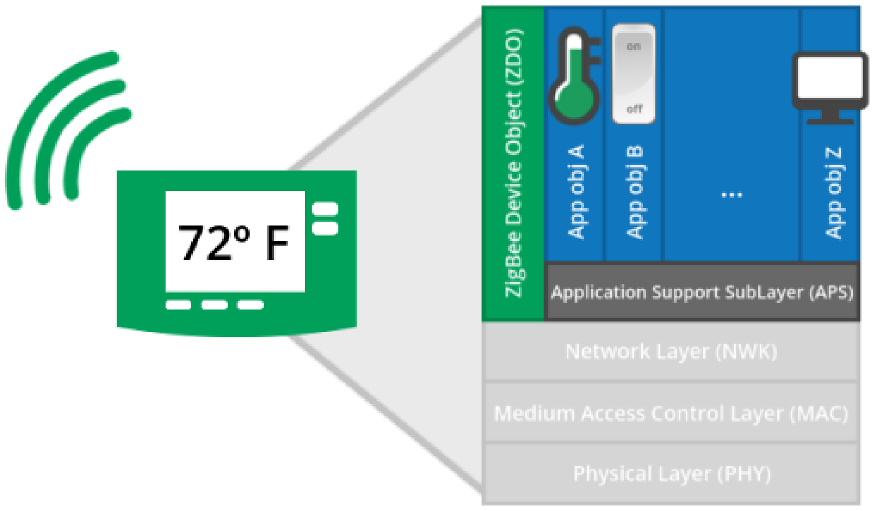
Each layer plays an important role in the transmission of wireless data, but it is the Application Layer that enables interoperability. This layer consists of the Application Support Sublayer (APS), the Zigbee Device Object (ZDO), and the Application Framework.

- **Application Support Sublayer**: provides an interface between the network layer and the application layer. It defines standardized messaging for specific tasks which support communication between devices from different manufacturers.
- **Application Framework**: environment in which application objects are hosted on Zigbee devices. In further discussions, this layer will be grouped with the APS sublayer.
- **Zigbee Device Object (ZDO)**: application object that provides device and service discovery features as well as advanced network management capabilities.

Zigbee Application Framework

A node in a Zigbee network is a radio device that controls and monitors multiple activities. Vendors can develop applications to manage these activities in the **Application layer** located on top of the Zigbee stack.

Each of the acitivites controlling or monitoring within the node is called an **application object**. This means a single physical device with a single Zigbee radio (node) may have several application objects with different purposes that are capable of controlling or monitoring different variables.


72° F
ZigBee Device Object (ZDO)
App obj A
App obj B
...
App obj Z
on
off
Application Support SubLayer (APS)
Network Layer (NWK)
Medium Access Control Layer (MAC)
Physical Layer (PHY)

For example, think of a thermostat. This product allows you to set the desired temperature, has a display, and can be turned on or off.

This thermostat is a single node on a Zigbee mesh with three application objects, each to perform a specific task:

- Heat or cool the room
- Display current temperature and other information
- Switch the unit on or off

Thermostats are manufactured by many different vendors. However, there is no guarantee that a thermostat from, for example, Acme Corp. will work with a heating/cooling unit from Stark Industries.

What makes the difference in Zigbee is the standardized technology and the interoperability. In fact, a Zigbee network is expected to have products from many vendors that all interoperate. This is probably one of the most important reasons to use Zigbee in a product.

Note The Zigbee Alliance certifies the stack and application profiles' compatibility to allow further interoperability between products developed by different vendors for a specific application.

The following concepts are explained in the next sections:

- Application profiles
- Clusters
- Endpoints
- Binding
- Node descriptors
- Zigbee Cluster Library

Application profiles

Zigbee defines application-level compatibility with application profiles. They describe how various application objects connect and work together, such as lights and switches, thermostats and heating units.

Application profiles specify a list of specific supported devices, device descriptions, and their features in terms of clusters. They contain an agreed-upon set of specific messages to allow applications from different vendors to interact.

For example, the Home Automation profile provides standard interfaces and device definitions to allow interoperability among Zigbee Home Automation devices produced by various manufacturers.

Note The use of an application profile allows further interoperability between the products developed by different vendors for a specific application.

Application profiles can be public or private:

- **Public profiles** are developed publicly by the Zigbee Alliance for use by manufacturers implementing products that need to work with products from other manufacturers.
- **Private profiles**, officially called Manufacturer Specific Profiles (MSP), are proprietary profiles developed privately by manufacturers. Applications that do not need to interact with other vendors' products use these profiles.

A 16-bit application **profile ID** identifies each application. Only the Zigbee Alliance can issue profile identifiers. If you want to build a custom profile, you must request a private profile ID from Zigbee Alliance.

Public profiles include:

- Home Automation (HA)
- Commercial Building Automation (CBA)
- Smart Energy (SE)
- Light Link (LL)
- Telecommunications Applications (TA)
- Personal Home and Health Care (PHHC)

Public profile specifications are available at http://www.Zigbee.org.

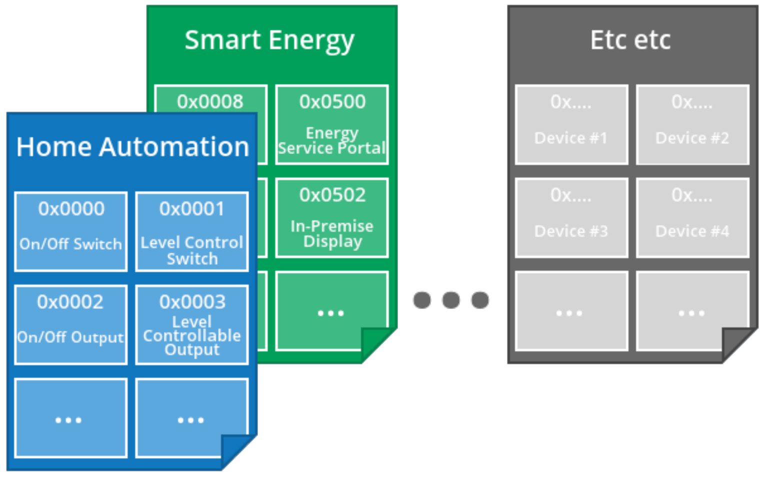
For our thermostat product, we can use public profiles to ensure compatibility between Acme and Stark Industries products and private profiles to define specific functionality that does not need to work with other vendors' products:

- Home Automation (HA) profile to control the temperature and enable it switching it on or off
- Smart Energy (SE) profile for the implementation of an In-Home display
- Proprietary profile or Manufacturer Specific Profiles (MSP) to implement custom functionality that checks the proper operation of the product

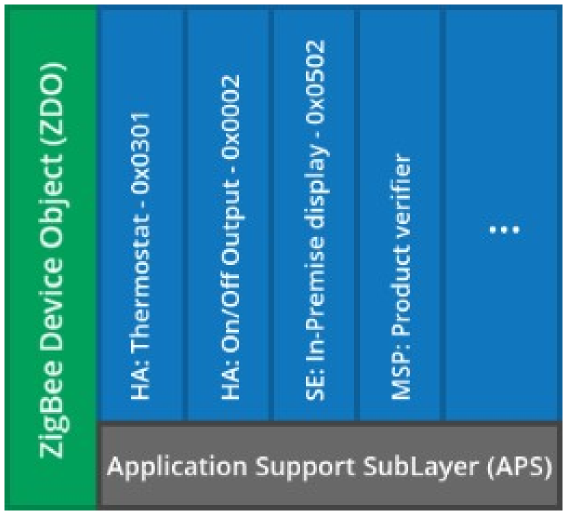
Device descriptions

Each Zigbee profile contains a specific list of **device descriptions** that describe the types of devices the profile supports. Each device description is identified by a unique 16-bit value called a **device ID**.

These devices have their own capabilities regarding what they can do, what kind of properties they have, and what kind of messages they send and receive. For example, the Home Automation profile describes devices such as an on/off switch, a light sensor, a thermostat, or a dimmable light that can be part of a HA network.



Note Device descriptions detail the behavior of each device in the application profile specification.


ZigBee Device Object (ZDO)
HA: Thermostat - 0x0301
HA: On/Off Output - 0x0002
SE: In-Premise display - 0x0502
MSP: Product verifier
...
Application Support SubLayer (APS)

The thermostat to implement four different device descriptions:

- An HA Thermostat with device ID 0x0301 allows to control the temperature.
- The HA On/Off Output (0x0002) is capable of being switched on and off.
- The SE In-Premise display device description, device ID 0x0502, shows information about energy consumption or price.
- Our proprietary profile defines a device to check the product operation, such as communication status, display functionality, and so on.

Clusters

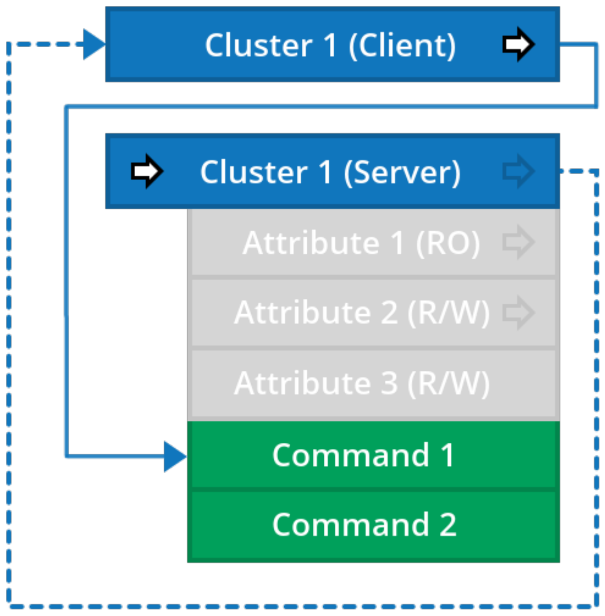
Each application object is associated with an application profile and contains **clusters** identified by a unique 16-bit **cluster ID**. Clusters consist of a set of properties and message types related to a certain function that define communication between application objects, like interfaces for features and domains.

Application profiles describe mandatory clusters and optional clusters for each device included in the profile.

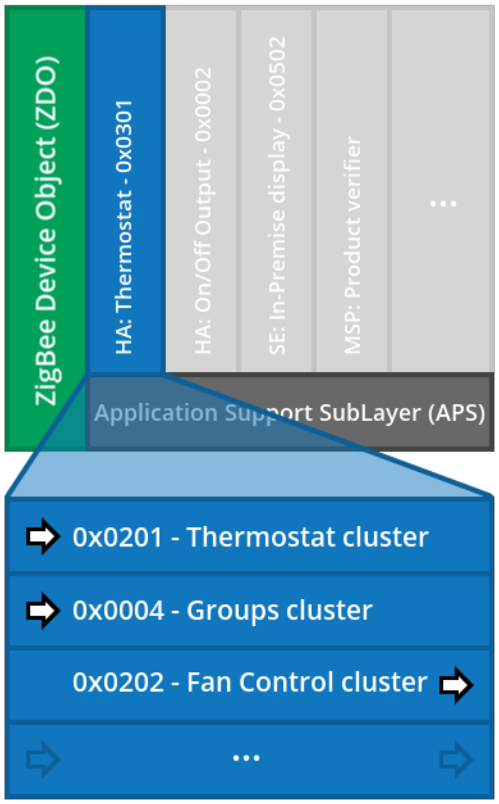
For example, we need to implement the HA thermostat (device ID: 0x0301) to develop our product. This device description includes:

- Mandatory cluster: Thermostat cluster (cluster ID: 0x0201).
- More optional clusters, such as Groups (0x0004) for group addressing, Fan Control (0x0202) to control the speed of a fan, or Temperature Measurement (0x0402) to receive temperature reports.

So, we must implement the Thermostat cluster (0x0201), and other optional clusters can be included as well depending on the features of the final product, such as Fan Control (0x0202) or Temperature Measurement (0x0402).



Each cluster has a collection of properties, called **attributes**, which is maintained on the device. This collection is also identified by a 16-bit value called an attribute identifier. The cluster also contains a set of **commands** that the device is responsible for sending or receiving.


ZigBee Device Object (ZDO)
HA: Thermostat - 0x0301
HA: On/Off Output - 0x0002
SE: In-Premise display - 0x0502
MSP: Product verifier
...
Application Support SubLayer (APS)
0x0201 - Thermostat cluster
0x0004 - Groups cluster
0x0202 - Fan Control cluster
...

For example, the Thermostat cluster (0x0201) supports several attributes to represent information, such as the LocalTemperature attribute (0x0000) to display the temperature in degrees Celsius. This cluster also includes the Setpoint Raise/Lower command (0x00) to increase or decrease the temperature by the specified amount.

Note A cluster is a set of commands and attributes. The Zigbee Cluster Library specifies common clusters that can be used by public or private profiles.

Clusters implement a client/server model and are directional:

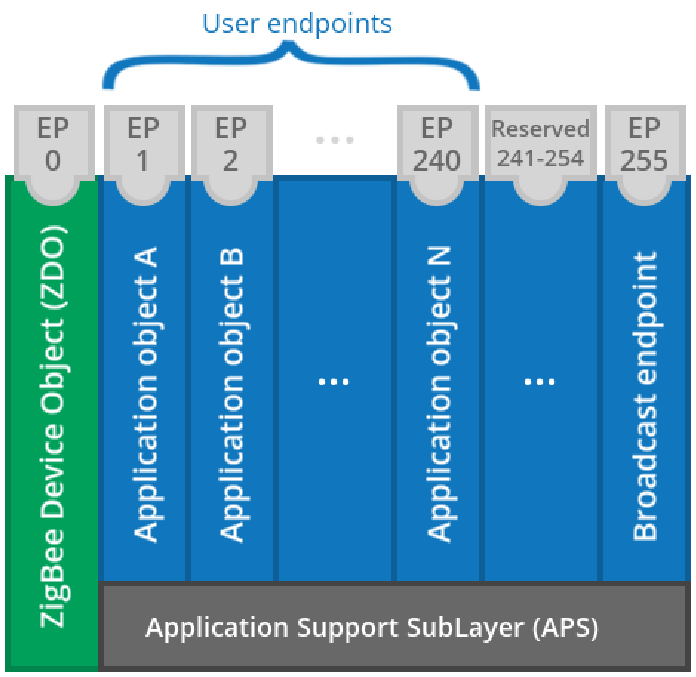
- **Outbound**: The **client** is the side that initiates the transaction and sends the message to the server. For example, a switch that sends an on/off command.
- **Inbound**: The **server** is the side that actually contains the attributes and performs the work. For example, a light that knows if it is on or off and completes the transaction by turning on/off when a command from the switch (the client) arrives.

Endpoints

We know a Zigbee node may have several application objects running on it. That is, our thermostat product, which is a node in the Zigbee network, is controlling the temperature and showing some information in its display, each of which is an application. But, how do we send some information to be shown to the display application object within the node? How do we route messages arriving at the node to the appropriate application?

Each application on the node must be uniquely identified. We do this with **endpoints**. Endpoints are service points and define each application object running in the Zigbee node.

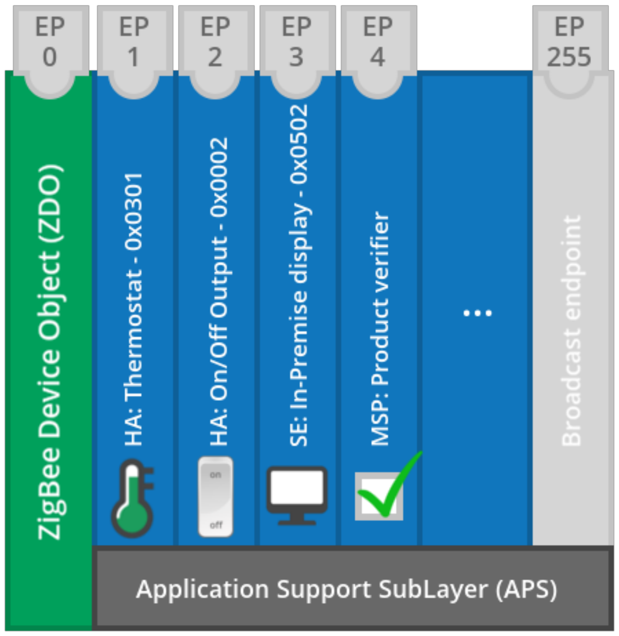
Note Endpoints describe different applications that are supported by a single radio. An endpoint is the "address" of a single application object within a node in the network.



The endpoint address is a user-defined 8-bit value, so there can be up to 255 endpoints defined within a node. However some endpoints are reserved:

Endpoints	Description
0	Built-in by the stack and reserved for Zigbee Device Object (ZDO) for network configuration and administration
1-240	Can be used for user applications
241-254	Reserved for special functions and can only be used for Zigbee Alliance-approved applications
255	Broadcast endpoint. The same data can be sent to all applications (endpoints) on a node by sending the message to this endpoint address

Note Each endpoint implements a single device description from a single application profile. Endpoints 1-240 can be allocated by users for any required application object. Different endpoints on a single node may represent devices from different application profiles.


EP 0
EP 1
EP 2
EP 3
EP 4
EP 255
ZigBee Device Object (ZDO)
HA: Thermostat - 0x0301
HA: On/Off Output - 0x0002
SE: In-Premise display - 0x0502
MSP: Product verifier
...
Broadcast endpoint
Application Support SubLayer (APS)

As an example, our thermostat product defines four user-defined endpoints besides ZDO and broadcast:

- Endpoint 0: ZDO endpoint
- Endpoint 1: Home Automation endpoint acting as a thermostat device (0x0301)
- Endpoint 2: Home Automation endpoint acting as an On/Off output (0x0002)
- Endpoint 3: Smart Energy endpoint acting as an In-Premise display (0x0502)
- Endpoint 4: Proprietary endpoint under a MSP (Manufacturer Specific Profile)
- Endpoint 255: Broadcast endpoint

Binding

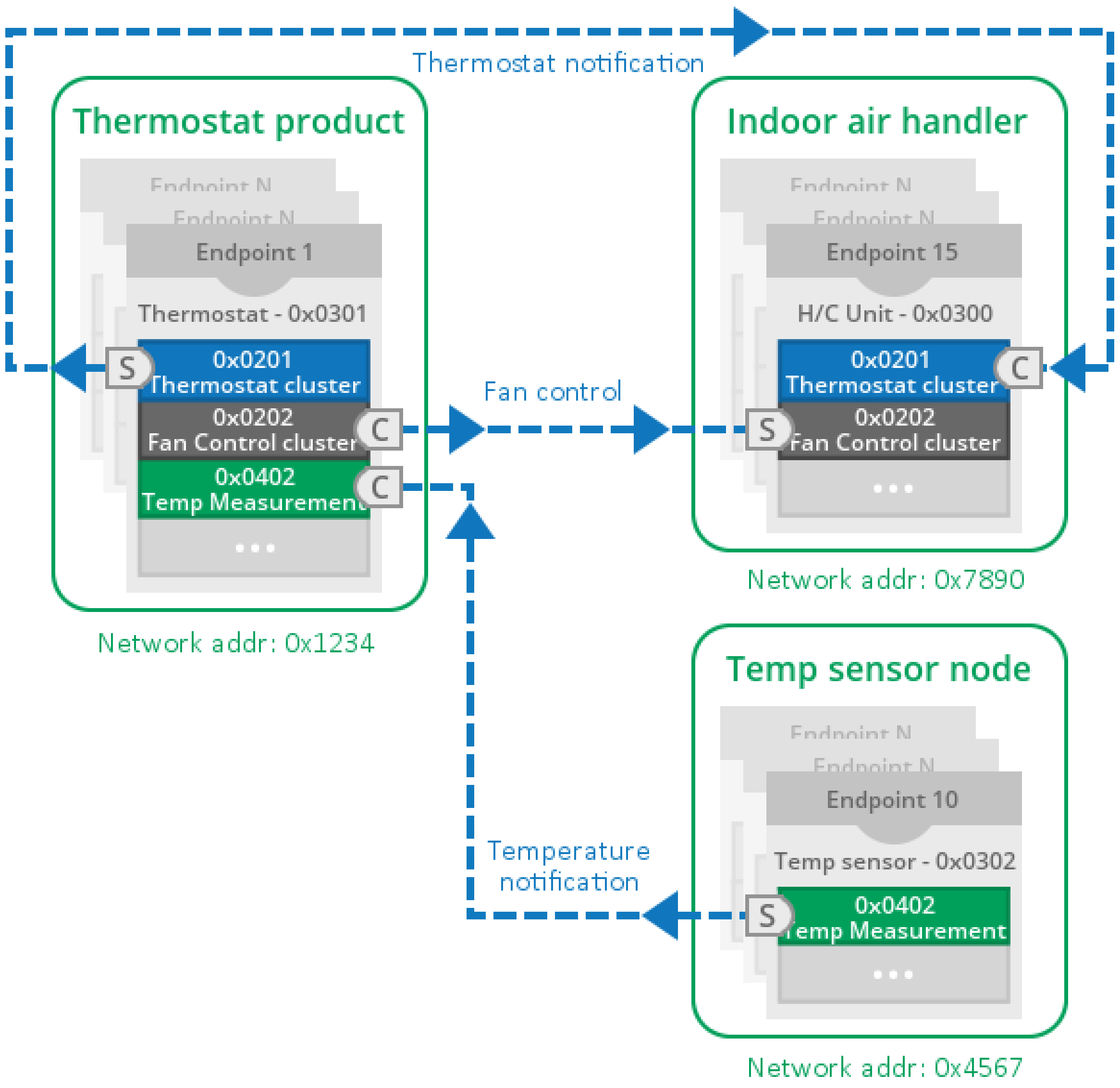
At a high level, binding is the process of creating logical links between application objects that are related and can communicate. For example, a temperature sensor sends its data to a thermostat. Devices (application objects) logically related in this way are called bound devices.

Binding is unidirectional. The sender (the temperature sensor) is "bound" to the receiving device (the thermostat), but the thermostat is not bound to the temperature sensor. Binding can also allow an endpoint on a node to be connected to one or more endpoints on one or more nodes.

Note Binding is a mechanism to establish virtual connections between endpoints on different nodes.

At a more detailed level, the binding mechanism associates information-generating applications with applications that can use the information.

The information is exchanged as clusters: two applications must have compatible clusters to be bound. For example, to create a binding between two applications on different nodes for controlling the temperature, one of those applications must be able to generate an output cluster related to temperature (the temperature sensor), and the other must be able to consume it as an input cluster (the thermostat).



The information regarding these logical links is stored in a binding table. Each entry contains the following information:

- Source endpoint of the sender application.

  In the example, the application object that measures the temperature is located at endpoint 10.
- Cluster ID of the information being transmitted between the applications. In the table, the Temperature Measurement (0x0402) cluster is the one that generates the data, a temperature value, to be transmitted.

Example: Temperature sensor binding table

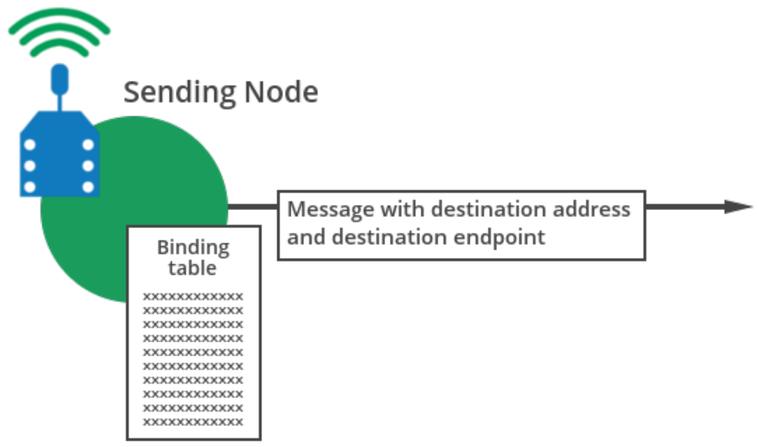
Source endpoint	Cluster ID	Destination address	Destination endpoint
10	0x0402	0x1234	1
...	...	...	...

- Destination network address and endpoint of the receiving application. Temperature data (source endpoint 10, cluster ID 0x0402) is going to be transmitted to the endpoint 1 of the network node 0x1234.

Depending on where the binding information (the entries of the binding table) is stored, binding is direct or indirect.

Direct binding

Direct binding, also called local binding, happens when the sending node stores the binding table locally (the temperature sensor).

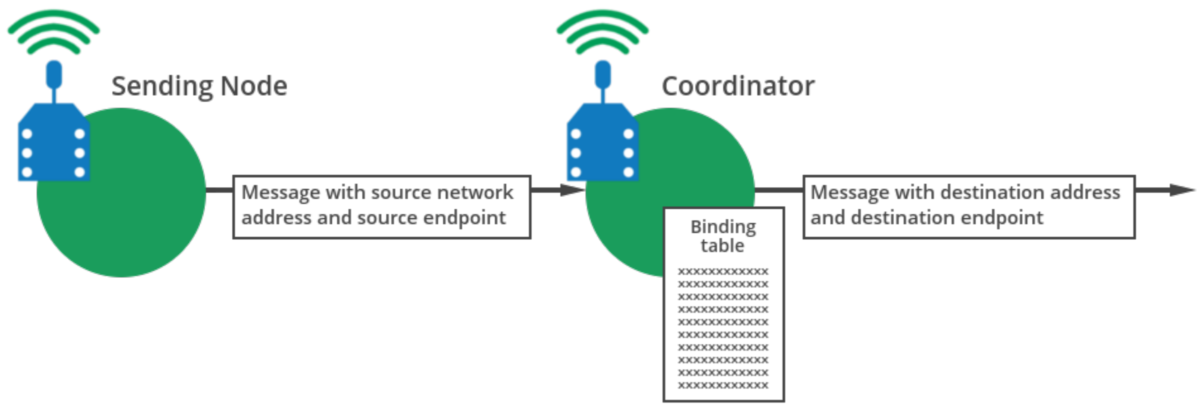


For instance, the temperature sensor node generates a new temperature value. The application object that measures the temperature is located at endpoint 10 and the cluster ID in charge of this is 0x0402. So, the sensor node looks for the entries in its binding table that match source endpoint 10 and cluster ID 0x0402 and generates a message for each one. In the example, there is only one match, so the sending node generates and transmits a single message with destination 0x1234 (the thermostat node) and destination endpoint 1.

Indirect binding

Indirect binding, also known as coordination binding, occurs when the binding table is stored on the coordinator node. This means all the messages must be sent via the coordinator.

When sending application data, the node transmits this data to the coordinator along with the source endpoint and cluster ID where it was generated and its own network address. The coordinator looks in the binding table for the entries that match the received source endpoint and cluster ID to create a message for each entry found, replicating the received data. Adding the destination address and destination endpoint stored in the binding entries completes the message. Finally, the coordinator transmits the messages directly to the right application in the destination node.


Sending Node
Message with source network address and source endpoint
Coordinator
Binding table
Message with destination address and destination endpoint

For instance, the temperature sensor node with address 0x4567 generates a new temperature value. The application object that measures the temperature is located at endpoint 10 and the cluster ID in charge of this is 0x0402. So, the sensor node transmits to the coordinator the new temperature, the source endpoint (10) and the cluster ID (0x0402) generating the value along with its own network address (0x4567). The coordinator looks for the entries in its binding table that match source endpoint 10 and cluster ID 0x0402 and generates a message for each one. In the example, there is only one match so the sending node generates and transmits to 0x1234 (the thermostat node), destination endpoint 1, a single message with the temperature value, source endpoint, and cluster ID.

Binding types

Binding is possible between a single output and input cluster but it can also be destined for groups of nodes or even multiple destinations by having multiple entries for the same source endpoint and cluster ID in the binding table. The types of bindings are:

- **One-to-one**: An endpoint binds to one (and only one) other endpoint.
- **One-to-many**: A source endpoint binds to more than one destination endpoint.
- **Many-to-one**: More than one source endpoint binds to a single destination endpoint.

Node descriptors

Other than the reserved endpoints, endpoint numbers are not standardized. Different manufactures may choose different endpoints for their applications. In other words, our product implements an on/off output on endpoint 2, but Acme Corp. uses endpoint 55 for the same on/off output on their products.

This means that when a node enters a network, it needs to query the rest of nodes to find out what endpoints they have and what services are implemented on those endpoints. Each endpoint returns some information when it is queried, such as which profile is implementing, which device description within that profile is being implemented, and the list of inbound (server) and outbound (client) clusters this endpoint has.

Every Zigbee node in the network defines several **descriptors** to identify itself and its endpoints:

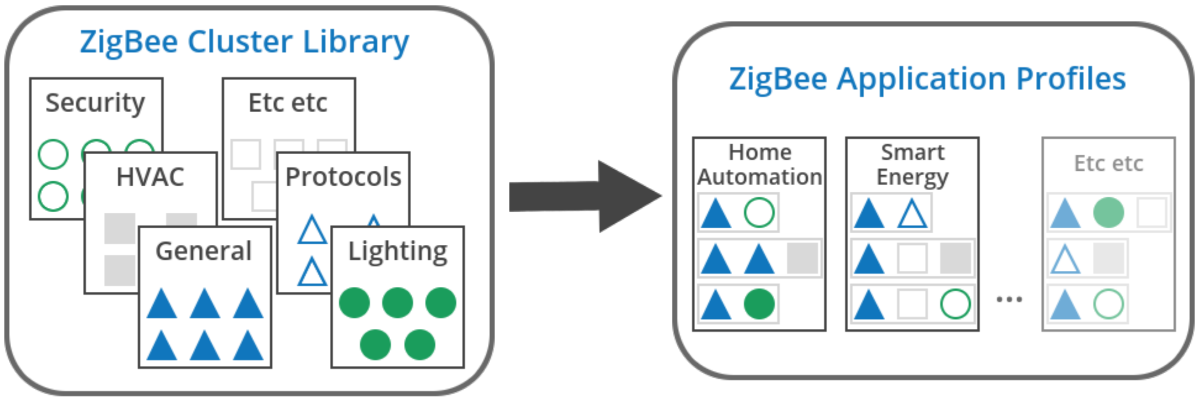
- **Node descriptor** contains information about the capabilities of the Zigbee node and is mandatory for each node. There is only one node descriptor in a node.
- **Power descriptor** gives a dynamic indication of the power status of the node and is mandatory for each node. There is only one node power descriptor in a node.
- **Simple descriptor** contains information specific to each endpoint contained in this node. The simple descriptor is mandatory for each endpoint present in the node.

- **Complex descriptor** contains extended information for each of the device descriptions contained in this node. The use of the complex descriptor is optional.
- **User descriptor** contains information that allows the user to identify the device using a user-friendly string. The use of the user descriptor is optional.

Zigbee Cluster Library

The Zigbee Cluster Library (ZCL) is a set of common clusters and cross-cluster commands used in application profiles.

The Zigbee Cluster Library groups the clusters in **functional domains**, such as General, Measurement and sensing, and Lighting. Clusters from these functional domains are used in the Zigbee Public Profiles to produce device descriptions, such as an on/off switch, a thermostat, or a dimmable light. Each Public Profile may also define its own specialized clusters, such as the Smart Energy Price Cluster.



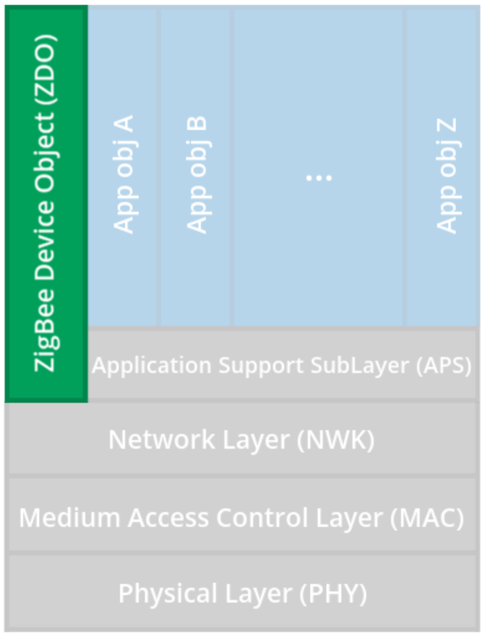
The Zigbee Cluster Library is released under its own specification, separate from the Zigbee specification. Go to http://www.Zigbee.org to download the ZCL specification.

Zigbee Device Object (ZDO)

The **Zigbee Device Object (ZDO)** is an application object responsible for initializing the Application Support Sublayer (APS), Network Layer (NWK), and Security Service Provider (SSP). It runs on the reserved endpoint 0 in every Zigbee device.

ZDO is typically required when developing a Zigbee product that interoperates in a public profile such as Home Automation or Smart Energy, or when communicating with Zigbee devices from other vendors. The ZDO can also be used to perform several management functions such as frequency agility, discovering routes and neighbors, and managing device connectivity.

The application profile defined for ZDO is the **Zigbee Device Profile (ZDP)**, which has an application profile identifier of 0x0000. ZDP is a management and discovery service layer supported on all Zigbee devices. Like all other profiles, it defines a set of services that perform a variety of advanced network management and device and service discovery options.


ZigBee Device Object (ZDO)
App obj A
App obj B
...
App obj Z
Application Support SubLayer (APS)
Network Layer (NWK)
Medium Access Control Layer (MAC)
Physical Layer (PHY)

ZDP services include the following features:

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

Each service has an assigned cluster ID, and most service requests have an associated response. In those cases, the client device makes a request, and then the server device sends the response back to the client device. The cluster ID for the response is exactly the same as the cluster ID for the request, but with the high bit set. For example, the Network Address Request is cluster 0x0000, and the Network Address Response is 0x8000.

The following table describes some ZDP services:

Cluster Name	Cluster ID	Description
Network Address Request	0x0000	Request a 16-bit address of the radio with a matching 64-bit address
Network Address Response	0x8000	Response that includes the 16-bit address of a device
LQI Request	0x0031	Request data from a neighbor table of a remote device
LQI Response	0x8031	Response that includes neighbor table data from a remote device

Cluster Name	Cluster ID	Description
Routing Table Request	0x0032	Request to retrieve routing table entries from a remote device
Routing Table Response	0x8032	Response that includes routing table entry data from a remote device
Active Endpoints Request	0x0005	Request a list of endpoints from a remote device
Active Endpoints Response	0x8005	Response that includes a list of endpoints from a remote device

For a detailed description of these and other ZDP services, read Supporting ZDOs with the XBee API or refer to the Zigbee specification.

Explicit Addressing frames

XBee devices support communication between Zigbee application objects running on the same node or among objects running on different nodes in a network.

This wireless data transmission is only possible if the modules are working in API operating mode and use the frames that include application addressing (endpoints, cluster ID, profile ID):

- Explicit Addressing Command (0x11)
- Explicit Rx Indicator (0x91)

Explicit Addressing Command frame

The Explicit Addressing Command (0x11) frame allows Zigbee application layer fields (endpoint, profile, and cluster ID) to be specified for a wireless data transmission.

In other words, an Explicit Addressing frame causes the module to send data wirelessly to the specified remote node, and to its proper application object specified by the destination endpoint, cluster ID, and profile ID. These frames allow you to send Zigbee Device Profile (ZDP), Zigbee Cluster Library (ZCL), and application profile commands to devices in the network.

This frame type includes specific fields for:

- Node addressing: 64-bit and 16-bit destination address.
- Service addressing: Source and destination endpoints, cluster ID, and profile ID.
- ZDP, ZCL, or application profile command in its data payload.

Frame parameter	Description
Frame ID	Identifies the data frame for the host to correlate with a subsequent Transmit Status (0x8B) frame. Setting Frame ID to '0' disables the response frame.

Frame parameter	Description
64-bit destination address	Set to the 64-bit address of the destination XBee module. The following addresses are also supported: 0x0000000000000000 - Coordinator address. 0x000000000000FFFF - Broadcast address. 0xFFFFFFFFFFFFFFFF - Unknown address if the destination's 64-bit address is unknown.
16-bit destination address	Set to the 16-bit address of the destination XBee module, if known. The following addresses are also supported: 0x0000 - Coordinator address. 0xFFFE - Unknown address if the destination's 16-bit address is unknown, or if sending a broadcast.
Source endpoint	Endpoint of the source that initiated the transmission.
Destination endpoint	Endpoint of the destination where the message is addressed.
Cluster ID	Cluster ID where the message was addressed.
Profile ID	Profile ID where the message was addressed.
Broadcast radius	0x00 – Maximum hops value.
Transmit options	Bitfield of supported transmission options. Supported values include the following: 0x01 - Disable retries. 0x04 - Indirect Addressing. 0x08 - Multicast Addressing. 0x20 - Enable APS encryption (if EE = 1). 0x40 - Use the extended transmission timeout for this destination. All unused and unsupported bits must be set to 0.
Data payload	Up to 255 bytes of data sent to the destination XBee moduleand the configured destination endpoint and cluster ID. For ZDP, ZCL, or application profile commands, all multi-byte parameter values must be sent in little endian byte order.

Explicit Rx Indicator frame

The Explicit Rx Indicator (0x91) frame contains the data wirelessly received and the application object addressing: source and destination endpoints, cluster ID and profile ID.

This frame helps when working with ZDP, ZCL, or application profile commands, since it indicates the application object information that generated the data inside, and the destination.

This frame type includes specific fields for:

- Node addressing: 64-bit and 16-bit source address
- Service addressing: Source and destination endpoints, cluster ID, and profile ID.
- ZDP, ZCL, or application profile response in its data payload.

Frame parameter	Description
64-bit source address	64-bit address of sender. Set to 0xFFFFFFFFFFFFFFFF (unknown 64-bit address) if the sender's 64-bit address is unknown.
16-bit source address	16-bit address of sender.
Source endpoint	Endpoint of the source that initiated the transmission.
Destination endpoint	Endpoint of the destination where the message is addressed.
Cluster ID	Cluster ID where the message was addressed.
Profile ID	Profile ID where the message was addressed.
Received options	Bitfield of supported transmission options. Supported values include the following: 0x01 - Packet Acknowledged. 0x02 - Packet was a broadcast packet. 0x20 - Packet encrypted with APS encryption. 0x40 - Packet sent with extended timeout enabled.
Data payload	Up to 255 bytes data received from the source XBee. For ZDP, ZCL, or application profile commands, all multi-byte parameter values are in little endian byte order.

Data payload format

The data payload field of the Explicit Addressing and Explicit Rx Indicator frames must follow the frame structure defined for ZDP and ZCL frames.

Note All multi-byte values must be sent/received in little endian byte order.

ZDP frame structure

Transaction sequence number	Transaction data	Transaction sequence number	Transaction data
1	2	...	n
A single byte to identify the ZDP transaction so a response frame can be related to the request frame.	A variable length field containing data for the individual ZDP transaction. Its format and length depends on the command being transmitted.		

ZCL frame structure

ZCL Header	ZCL Payload	ZCL Header	ZCL Payload	ZCL Header	ZCL Payload	ZCL Header
Frame control	**Manufacturer code**	**Transaction sequence number**	**Command ID**	**Frame payload**	**Frame control**	**Manufacturer code**
1	2	3	4	5	...	6
Byte to define the command type and other flags.	Two bytes field specifying the Zigbee manufacturer code.	Byte to identifying the transaction so a response frame can be related to the request frame.	Byte to specify the cluster command.	A variable length field with data specific to individual command types.		

For more information about the frames structure, refer to the Zigbee specification and the Zigbee Cluster Library available at http://www.Zigbee.org.

Receive Zigbee commands and responses

To receive ZDP, ZCL, and application profile commands and responses:

- XBees modules must work in API mode, that is AP must be set to API enabled [1] or API enabled with escaping [2].
- The API Output Mode (AO) setting must be set to Explicit [1] or Explicit with ZDO Passthru [3].

The Explicit [1] output mode (AO = 1) enables the explicit receive API frame (0x91) which indicates the source and destination endpoints, cluster ID, and profile ID.

The Explicit with ZDO Passthru [3] mode (AO = 3) is similar to the Explicit [1] mode, but it also outputs some ZDO packets received by the XBee module through the serial interface as explicit data to be responded by external processors:

- ZDO requests that are not supported by the stack.
- ZDO Simple descriptor request (Cluster ID: 0x0004).
- ZDO Active endpoints request (Cluster ID: 0x0005).
- ZDO Match descriptor request (Cluster ID: 0x0006).

For an XBee to received ZDP, ZCL, or application profile responses:

- API mode (AP) must be enabled (AP = 1 or AP = 2).
- AO parameter must be set to Explicit [1] or Explicit with ZDO Passthru [3] to enable the explicit receive API frame.

Examples: explicit data and ZDO

These examples illustrate how to use the Explicit Addressing Command Frame (0x11) to work with profiles, clusters, and endpoints. Both examples request the neighbor table of a remote node, but the procedure is different. First, you use the XCTU console to create the packet and analyze the response. Then you use a Java application and the XBee Java Library to abstract the process of sending and receiving the packets.

Use the Zigbee Device Object (ZDO) to obtain the neighbors of a remote node. You can use a similar approach for other profiles and cluster IDs by adjusting the request to send, the parse of the received data, and the actions to perform depending on what is received.

In the remote node, the ZDO is the application object that manages the neighbor list. It is located at endpoint 0 and under the Zigbee Device Profile (ZDP). The request must be addressed to the cluster of this object that is able to return a neighbor's table, the LQI Request (cluster ID: 0x0031).

On receipt, the remote node's ZDO generates an answer from the 0x8031 cluster (LQI Response) addressed to the same application object, ZDO at endpoint 0, of the requester node.

	Request to remote node		**Response from remote node**	
Source endpoint	ZDO	0	ZDO	0
Destination endpoint	ZDO	0	ZDO	0
Profile ID	ZDP	0x0000	ZDP	0x0000
Cluster ID	LQI Request	0x0031	LQI Response	0x8031

Example: obtain the neighbor table using the XBee Java Library

In this example, you use XCTU to send an LQI (neighbor table) Request from the coordinator to the router.

Looking at the Zigbee specification, the cluster ID for a LQI Request is 0x0031, and the payload only requires that you specify the start index in the neighbor table. The response cannot include more that 2-3 entries, so multiple LQI requests may be required to read the entire table.

If you get stuck, see Troubleshooting.

Step 1: Requirements

Hardware

- Two XBee Zigbee modules
- Two XBee Grove Development Boards
- Two micro USB cables
- One computer, although you may also use two

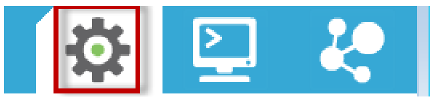
Software

- XCTU 6.3.1 or later

Step 2: Connect the components

To get started, connect the components and start XCTU.

1. Plug the XBee modules into the XBee Grove Development Boards and connect them to your computer using the micro USB cables provided. For more information, see Plug in the XBee module.
2. After connecting the modules to your computer, open XCTU.
3. Make sure you are in **Configuration working mode**.



Step 3: Configure the XBee devices

For this example, two modules are sufficient. The coordinator send the request to read the neighbor table of the router. Before creating and sending the frame, configure the XBees as follows:

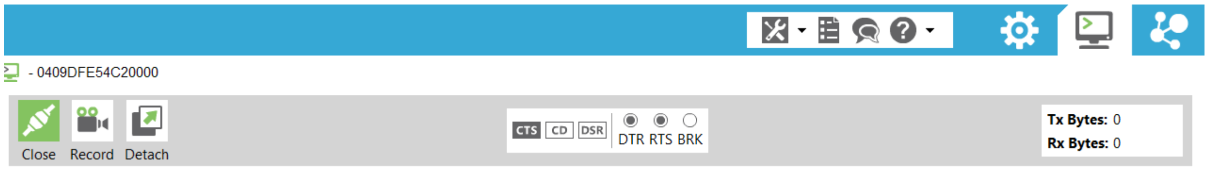
Param	XBee A	XBee B	Effect
ID	2015	2015	Defines the network that a radio will attach to. This must be the same for all radios on your network.
JV	—	Enabled [1]	Verifies if a coordinator exists on the same channel to join the network or to leave if it cannot be found.
CE	Enabled [1]	—	Sets the device as coordinator.

Param	XBee A	XBee B	Effect
NI	COORD	ROUTER	Defines the node identifier, a human-friendly name for the module. ⚠ The default NI value is a blank space. Make sure to delete the space when you change the value.
AP	API enabled [1]	API enabled [1]	Enables API mode.
AO	Explicit [1]	—	Enables the explicit receive API frame.

Note The dash (**—**) in the table means to keep the default value. Do not change the default value.

Step 4: Open the XCTU console

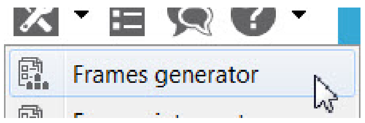
1. Switch to the **Consoles working mode** .
2. Open the serial connection with the COORD module .



Step 5: Generate the frame

Next, generate the Explicit Addressing Command Frame in the COORD console. To create it, follow these steps:

1. Click **Add new frame to the list** .
2. Open the **Frames Generator** tool.

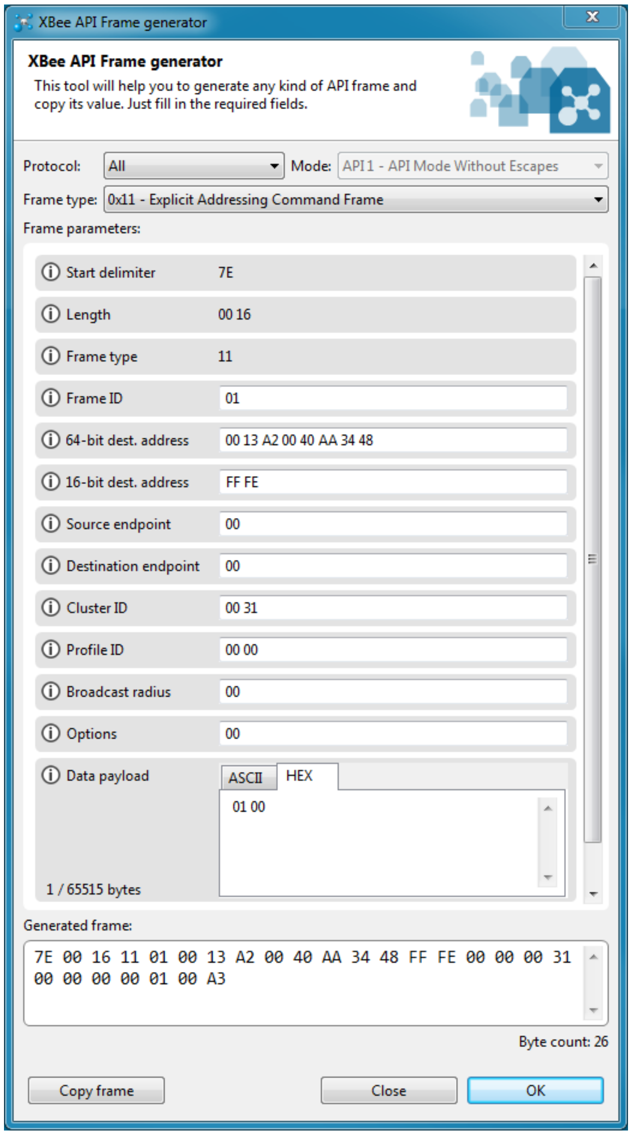


3. In the **Frame type** section, select 0x11 - Explicit Addressing Command Frame.
4. In the **64-bit dest. address** box, type 0013A200XXXXXXXX, the 64-bit address of ROUTER.

   In the **Source endpoint** box, type 00.
5. In the **Destination endpoint** box, type 00.

6. In the **Cluster ID** box, type 0031.
7. In the **Profile ID** box, type 0000.
8. In the **Data payload**, click the HEX tab and type 0100.

01	00
Sequence number	Start index


XBee API Frame generator
XBee API Frame generator
This tool will help you to generate any kind of API frame and copy its value. Just fill in the required fields.
ASCII
HEX
Copy frame
Close
OK

Protocol: All
Mode: API 1 - API Mode Without Escapes
Frame type: 0x11 - Explicit Addressing Command Frame
Frame parameters:

Start delimiter	7E
Length	00 16
Frame type	11
Frame ID	01
64-bit dest. address	00 13 A2 00 40 AA 34 48
16-bit dest. address	FF FE
Source endpoint	00
Destination endpoint	00
Cluster ID	00 31
Profile ID	00 00
Broadcast radius	00
Options	00
Data payload	01 00

1 / 65515 bytes

Generated frame:

```
7E 00 16 11 01 00 13 A2 00 40 AA 34 48 FF FE 00 00 00 31
00 00 00 00 01 00 A3
```

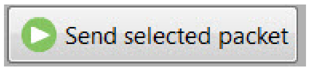
Byte count: 26

9. Click **OK**.
10. Click **Add frame**.

Step 6: Send the command frame

After you have created an Explicit Addressing Command Frame, you must send it to the local XBee module.

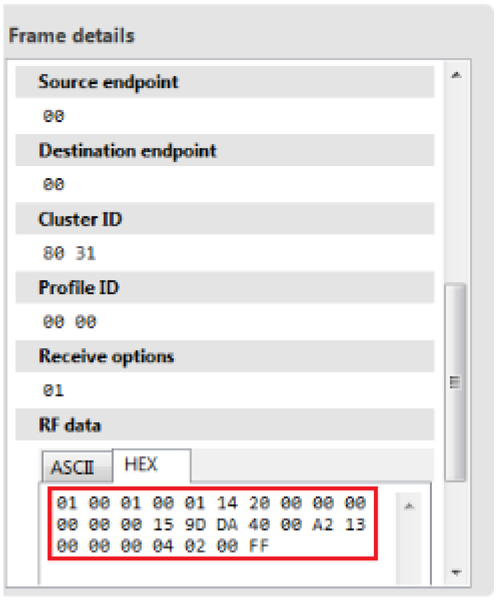
1. Select the frame in the XCTU **Send frames** section.
2. Click **Send selected packet**.



The **Frames log** indicates that one frame has been sent (blue) and another has been received (red).

Step 7: Analyze the responses

Select the Explicit RX Indicator frame in the frames log to view the frame details on the right panel. Analyze each field, and check each of the following:



Source endpoint
00
Destination endpoint
00
Cluster ID
80 31
Profile ID
00 00
Receive options
01
RF data

```
01 00 01 00 01 14 20 00 00 00
00 00 00 15 9D DA 40 00 A2 13
00 00 00 04 02 00 FF
```

- **Frame type**: the received frame is an Explicit RX Indicator.
- **Source and Destination endpoints**: 00, the ZDO endpoint.
- **Cluster ID**: the same as the cluster ID for the request, but with the high bit set.
- **Profile ID**: the ID of the Zigbee Device Profile.
- **RF data**: the response that contains the list of neighbor table entries.

01	00	01	00	01	...
Sequence number	Status (OK)	Total number of entries	Start index	Number of entries in the response	List of entries

In this case, there is only one neighbor table entry:

Name	Size (bits)	Value	Description
Extended PAN ID	64	14 20 00 00 00 00 00 00	The 64-bit extended PAN ID of the neighboring device.
Extended Address	64	15 9D DA 40 00 A2 13 00	The 64-bit address of the neighboring device.
Network Address	16	00 00	The 16-bit address of the neighboring device.
Device Type	2	04	The type of neighbor: 0x0 – Zigbee coordinator 0x1 – Zigbee router 0x2 – Zigbee end device 0x3 – Unknown
Receiver On When Idle	2		Indicates if the neighbor's receiver is enabled during idle times: 0x0 – Receiver is off 0x1 – Receiver is on 0x2 – Unknown
Relationship	3		The relationship of the neighbor with the remote device: 0x0 – Neighbor is the parent 0x1 – Neighbor is a child 0x2 – Neighbor is a sibling 0x3 – None of the above 0x4 – Previous child
Reserved	1		Set to 0.
Permit Joining	2	02	Indicates if the neighbor is accepting join requests: 0x0 – Neighbor not accepting joins 0x1 – Neighbor is accepting joins 0x2 – Unknown
Reserved	6	Set to 0	
Depth	8	00	The tree depth of the neighbor device. A value of 0x00 indicates the device is the Zigbee coordinator of the network.
LQI	8	FF	The estimated link quality of data transmissions from this neighboring device.

The entry in the router's neighbor table corresponds to the coordinator of the network.

Once you have finished, click the **Close the serial connection** to disconnect the console.

Example: obtain the neighbor table using the XBee Java Library

Now that you have learned how to request the neighbor table by creating the packet and sending it with XCTU, you complete the same task, but with the help of the XBee Java Library.

The Java application creates the packet to request the neighbor table, send it to the other module, receive the response, parse it, and display the details of each neighbor.

If you get stuck, see Troubleshooting.

Step 1: Requirements

For this setup you need the following hardware and software.

Hardware

- Three XBee Zigbee Mesh Kit modules
- Three XBee Grove Development Boards
- Three micro USB cables
- One computer

Software

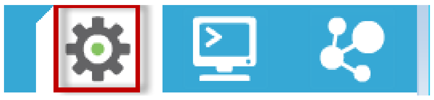
- XCTU 6.3.1 or later
- XBee Java Library (XBJL-X.Y.Z.zip release file)
- Java Virtual Machine 6 or later
- A Java IDE (such as Eclipse or NetBeans)

Tip For more information about XCTU, see the XCTU walkthrough.

Step 2: Connect the components

To get started, connect the components and start XCTU.

1. Plug the XBee modules into the XBee Grove Development Boards and connect them to your computer using the micro USB cables provided. For more information, see Plug in the XBee module.
2. After connecting the modules to your computer, open XCTU.
3. Make sure you are in **Configuration working mode**.



Step 3: Configure the XBee devices

Every XBee module has a different role and work in API mode.

1. Restore the default settings of all XBees with the Load default firmware settings.
2. Use XCTU to configure the following parameters:

Param	XBee A	XBee B	XBee C	Effect
ID	2015	2015	2015	Defines the network a radio will connect to. This parameter must be the same for all radios on your network.
JV	—	Enabled [1]	Enabled [1]	Verifies if a coordinator exists on the same channel to join the network or to leave if it cannot be found.
CE	Enabled [1]	—	—	Sets the device as coordinator.
NI	COORD	ROUTER_1	ROUTER_2	Defines the node identifier, a human-friendly name for the module. ⚠ The default NI value is a blank space. Make sure to delete the space when you change the value.
AP	API enabled [1]	API enabled [1]	API enabled [1]	Enables API mode.

Note The dash (**—**) in the table means to keep the default value. Do not change the default value.

3. Write the settings of all XBees with the **Write radio settings** button at the top of the Radio Configuration section.

To get the neighbors of a remote node, go to Step 4: Create a Java project project.

Step 4: Get neighbors: create a Java project

Create an empty Java project named using Eclipse or NetBeans with the following project name:**XBeeExplicitData**.

Option 1: Eclipse

a. Select **File > New**, and click the **Java Project**.
b. The **New Java Project** window appears. Enter the Project name.
c. Click **Next**.

or

Option 2: NetBeans

a. Select **File > New project....**
b. The New Project window appears. In the Categories frame, select **Java > Java Application** from the panel on the right, and click **Next**.
c. Enter the Project name and the Project Location. Clear the Create Main Class option; you will create this later.

d. Click **Finish** to create the project. The window closes and the project appears in the Projects view list on the left side of the IDE.

Step 5: Link libraries to the project

This topic describes how to link the **XBee Java Library**, the **RXTX library** (including the native one), and the **logger library** to the project.

1. Download the XBJL_X.Y.Z.zip library.
2. Unzip the XBJL_X.Y.Z.zip library.
3. Link the libraries using Eclipse or NetBeans:

Option 1: Eclipse

a. Go to the **Libraries** tab of the New Java Project window.
b. Click **Add External JARs....**
c. In the JAR Selection window, search the folder where you unzipped the XBee Java Library and open the **xbee-java-library-X.Y.Z.jar** file.
d. Click **Add External JARs...** again.
e. Go to the **extra-libs** folder and select the following files:
   - **rxtx-2.2.jar**
   - **slf4j-api-x.y.z.jar**
   - **slf4j-nop-x.y.z.jar**
f. Expand the **rxtx-2.2.jar** file of the Libraries tab list, select **Native library location**, and click **Edit....**
g. Click **External folder...** to navigate to the **extra-libs\native\Windows\win32** folder of the directory where you unzipped the XBee Java Library file (XBJL_X.Y.Z.zip).
   - Replace **Windows\win32** with the directory that matches your operating system and the Java Virtual Machine installed (32 or 64 bits). If you don't know which Java Virtual Machine is installed in your computer, open a terminal or command prompt and execute:

```
java -version
```

h. Click **OK** to add the path to the native libraries.
i. Click **Finish**.

or

Option 2: NetBeans

a. From **Projects** view, right-click your project and go to **Properties**.
b. In the categories list on the left, go to **Libraries** and click **Add JAR/Folder**.
c. In the **Add JAR/Folder** window, search the folder where you unzipped the XBee Java Library and open the **xbjlib-X.Y.X.ja**r file.
d. Click **Add JAR/Folder** again.
e. Go to the **extra-libs** folder and select the following files:

- **rxtx-2.2.jar**
- **slf4j-api- x.y.z .jar**
- **slf4j-nop- x.y.z .jar**

f. Select **Run** in the left tree of the **Properties** dialog.

g. In the **VM Options** field, add the following option:

```
-Djava.library.path=<path_where_the_XBee_Java_Library_is_unzipped>\extra-
libs\native\Windows\win32
```

where:

- **<path_where_the_XBee_Java_Library_is_unzipped>** is the absolute path of the directory where you unzipped the XBee Java Library file (XBJL_X.Y.Z.zip)
- **Windows\win32** is the directory that matches your operating system and the Java Virtual Machine installed (32 or 64 bits). If you don't know which Java Virtual Machine is installed in your computer, open a terminal or command prompt and execute:

```
java -version
```

h. Click **OK**.

Step 6: Add the source code to the project

Follow these steps to add the source to the project.

1. Download and unzip the ExplicitDataSample-src.zip file.
2. Add the Java source file with Eclipse or NetBeans.

Option 1: Eclipse

a. Copy the files inside the zip file.

b. In the **Package Explorer** view, **src** folder inside the project and right-click.

c. From the context menu, select **Paste**.

d. Double-click each copied file to open in the editor.

e. A line at the top of the pasted files is underlined in red. Click on that line; a pop-up appears. Select the first option (**Move 'MainApp.java' to package '...'**) to resolve the error.

or

Option 2: NetBeans

a. Copy the files inside the zip file.

b. In the **Projects** view, select the **Source Packages** folder inside the project and right-click.

c. From the context menu, select **Paste**.

d. Double-click each copied file to open in the editor.

e. A line at the top of the pasted files is underlined in red. Click on that line; a pop-up appears. Select the first option (**Move class to correct folder**) to resolve the error.

Step 7: Set the port name and launch the application

For this step, set the port name and launch the application.

1. Change the port name in the Java source code to match the port that the COORD module is connected to.

```
// TODO: Replace with the port where your coordinator module is connected
private static final String PORT = "COM1";
// TODO: Replace with the baud rate of your coordinator module.
private static final int BAUD_RATE = 9600
```

2. Launch the application on your computer.
3. When the prompt (>>) displayed after scanning the network, write the name of the remote node to get its neighbors. The output of the application should be similar to the following:

```
+-----------------------------------------------+
|              Explicit Data Sample             |
+-----------------------------------------------+
WARNING:  RXTX Version mismatch
Jar version = RXTX-2.2pre1
native lib Version = RXTX-2.2pre2
Local XBee: COORD
Scanning the network, please wait... Devices found.
Type the node name to retrieve its neighbor table and press <ENTER> (only
<ENTER> to finish):
- ROUTER_1
- ROUTER_2
>> ROUTER_1
Total number of neighbors: 2

Neighbor: 1
-----------------------------------
PAN ID:         0000000000002015
64-bit address: 0013A20040DA9D05
16-bit address: 0000
Device type:    Coordinator (0)
Relationship:   Parent (0)
Join requests:  Unknown (2)
Tree depth:     0
LQI:            255 / 255

Neighbor: 2
-----------------------------------
PAN ID:         0000000000002015
64-bit address: 0013A20040DA9D2B
16-bit address: 7EF4
Device type:    Router (1)
Relationship:   None of parent, child, sibling (3)
Join requests:  Unknown (2)
Tree depth:     15
LQI:            255 / 255

#################################################

Type the node name to retrieve its neighbor table and press <ENTER> (only
<ENTER> to finish):
- ROUTER_1
- ROUTER_2
>>
```

Step 8: Section summary of explicit data and ZDO

In this section, you learned the following:

- Zigbee offers a standardized technology and interoperability between products from many vendors. The Zigbee Alliance certifies the stack and application compatibility.
- A single physical Zigbee node may have several application objects with different purposes running on top of the Zigbee stack. Each application object running in a Zigbee node is "addressed" by an endpoint:
  - Each endpoint implements a single device description from a single application profile.
  - Endpoint 0 is reserved for Zigbee Device Object (ZDO).
  - Endpoints 1-240 can be allocated by users for any required application object.
- The Zigbee Device Object (ZDO) is the application responsible for network configuration and administration:
  - The application profile defined for ZDO is the Zigbee Device Profile (ZDP), which has an application profile identifier of 0x0000.
  - ZDP defines a set of services for advanced network management, device discovery, and service discovery options.
  - ZDP services are detailed in the Zigbee specification.
- The Explicit Addressing Command Frame (0x11) allows you to send commands to application objects including ZDO, ZCL and application profiles.
- To receive application objects requests or responses the API Output Mode (AO) setting must be set to Explicit [1] or Explicit with ZDO Passthru [3].

Step 9: Do more with explicit data and ZDO

If you are ready to move beyond this exercise and extend the example, try the following:

- Use different ZDO clusters to get the 16-bit or 64-bit address of a remote node, its routing table, or the descriptor of a specific endpoint.
- Extend the network by adding more XBee modules so you can request to more remote nodes and obtain bigger neighbor lists.
- Implement two different application objects from a public profile in different nodes and communicate between them. For example, from the Home Automation profile, develop an On/Off Light Switch on one node and a On/Off Light on the other so an LED of the light device is turned on when a button of the light switch device is pressed.
- Try to communicate with a Zigbee-certified product. For example, control a Zigbee light bulb using your XBees modules.

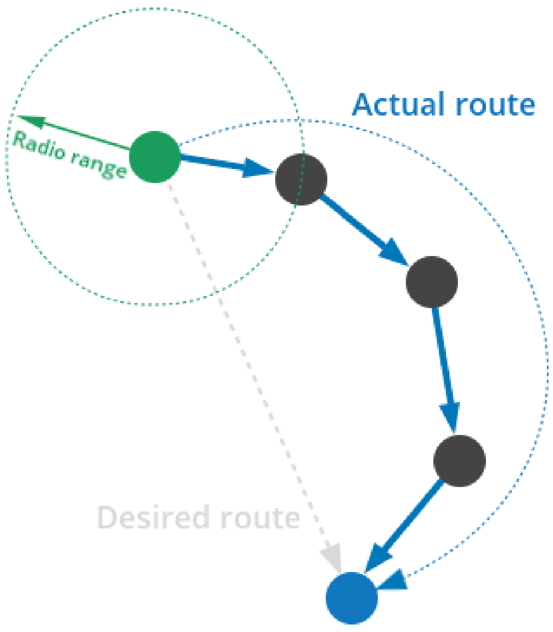
Large networks routing

The basic function of a network is to transfer data from one node to another. In the simplest data communication, the data is transmitted directly from the source node to the destination node. However, direct communication may not be possible if the two nodes are far apart or in a difficult environment.

In this case, it is necessary to send the data to another node within the radio range, which then passes it on to another node, and so on until the data reaches the desired destination node.

Routing is the process of receiving data destined for another node and passing it on. Each of the intermediary nodes between source and destination is called **hop**.

Note Routing allows the range of a network to be extended beyond the distances supported by direct radio communication.



A message is normally routed along an already discovered route. But if this route does not exist, the nodes involved in transmitting the data initiates a **route discovery**. Once completed, the message will be sent along the calculated route.

Note The route discovery process finds the best available route to the destination when sending a message.

The route discovery process is based on the **Ad-hoc On demand Distance Vector** routing (AODV) protocol. The AODV protocol uses tables in each node to store the next hop for a destination node. When a source node A must discover a route to a destination node B, route discovery involves the following steps:

1. Source node A sends a route request broadcast. The route request contains the source and destination network addresses and a path cost field to measure the route quality.
2. All routing nodes (coordinator and routers) eventually receives the broadcast. When a node receives this message:
   - It updates the path cost field.
   - It creates a temporary entry in its route discovery table.
   - It forwards the message.
3. As the reply travels back through the network, the hop count and a signal quality measure for each hop are recorded as described. Each routing node in the path can build a routing table entry containing the best path to the destination node B.
4. When the destination node B receives the route request:
   a. It compares the "path cost" field against previously received route requests.
   b. If the cost stored in the request is better than any previously received, the destination node, B, will transmit a route reply to the node that originated the request, A.
5. Eventually each routing node in the path will have a routing table entry and the route from source A to destination B is established. Note that the discovered route is unidirectional, the corresponding route from destination B to source A is not known.

In large networks where an XBee transmits data to many remote modules, AODV routing would require performing one route discovery for each destination node to establish a route. If there are more destinations than available routing table entries, established routes would be overwritten with new routes, causing route discoveries to occur more regularly. This could result in larger delays and decreased network performance.

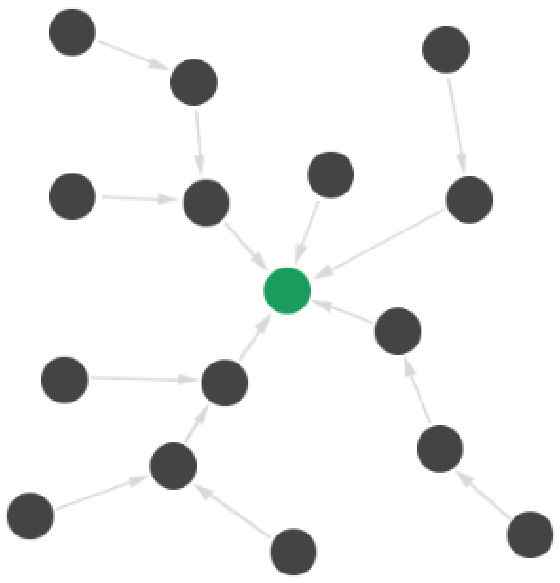
Many-to-one and source routing features address limitations in mesh network routing where table size requirements are large in certain wireless data transmission scenarios.

Many-to-one routing

A common scenario in a wireless network is when most of the nodes need to communicate with a single node that performs some centralized function. This node is often referred to as a collector or concentrator.

If every XBee in this kind of network had to discover a route before sending data to the collector, the network could easily become inundated with broadcast route discovery messages.

Many-to-one routing is an optimization for this scenario. Rather than require each XBee to perform its own route discovery, the data collector sends a single many-to-one broadcast transmission to establish reverse routes on all devices. Since no responses are generated, network traffic congestion is minimized.



Note Many-to-one routing establishes routing paths from many XBees to one data collector and allows any node in the network to route data to a well known concentrator through a single routing table entry in every device.

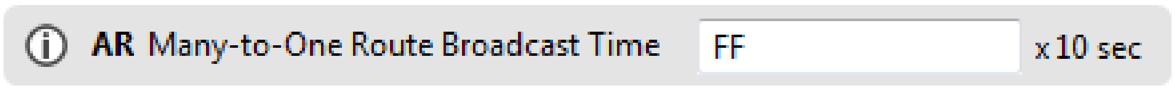
In a single many-to-one route discovery operation, the route to the data collector is established in all devices:

1. The data collector broadcasts a many-to-one route request message with the target discovery address set to the address of the data collector.
2. XBee modules receiving this request store a reverse many-to-one routing table entry to create a path back to the data collector.
3. The Zigbee stack on a device uses historical link quality information about each neighbor to select a reliable neighbor for the reverse route.
4. The many-to-one route request should be sent periodically to update and refresh the reverse routes in the network.

You can also use Many-to-one routing if there are multiple data collectors in the network. If more than one collector sends a many-to-one broadcast, XBee modules create one reverse routing table entry for each collector.

Enable many-to-one routing

Using many-to-one routing with an XBee-based central collector node is easy: the Many-to-one Route Broadcast Timeout (**AR**) parameter enables many-to-one broadcasting on an XBee module.



AR Many-to-One Route Broadcast Time FF x 10 sec

The **AR** parameter sets a time interval (measured in 10 second units) for sending the many-to-one broadcast transmission. Setting **AR** to 0xFF di sables many-to-one broadcasting on the device. Setting **AR** to 0 causes an XBee to immediately send a single a many-to-one broadcast.

Disable many-to-one routing

To disable many-to-one routing in a network:

1. Set **AR** parameter on the central data collector node to 0xFF and save the configuration. This ends the many-to-one broadcast sent by the data collector.
2. Broadcast a Software Reset (**FR**) command to the network and wait for the network to reform. This removes the data collector status as an aggregator from the routing tables.

Source routing

When a data collector node needs to communicate with the rest of nodes in the network and not just receive data from them, problems arise. The data collector, and even other nodes, would need enough routing table entries for every node in the network. Or, each node has to discover the route before sending data to the destination.

The maximum number of routing table entries in an XBee module is 40. This means networks containing more than 40 XBee modules could easily become inundated with broadcast route discovery messages.

Source routing allows the collector to store and specify routes for many remotes. The collector remembers the route taken by a message to the data collector so it can be retrieved later to indicate the route of a message from the collector to a certain remote:

Collector
0x000E
0x000D
0x000C
0x000B
0x000A

1. The data collector must send periodic many-to-one route request broadcasts, so many-to-one routes to the central collector are created on all remotes.
2. Every time a remote XBee module transmits data using a many-to-one route, it first sends a route record transmission.
3. The route record transmission is unicast along the many-to-one route until it reaches the data collector.

4. As the route record traverses the many-to-one route, each node in the route appends its own 16-bit address into the payload of the route record.
5. When the route record reaches the data collector, it contains the address of the sender and the 16-bit address of each hop in the route.
6. The data collector can store the routing information and retrieve it later to send a source routed packet to the remote module.

Note Source routing allows a data collector to route responses back to each device, supplying a many-to-one data request without additional route table entries.

Use source routing

To use source routing:

1. The data collector must work in API mode.
2. The collector must send periodic many-to-one route request (**AR** parameter value must be different from 0xFF).
3. You need a microcontroller or a PC connected to the data collector to automatically capture and store the routes to the rest of network nodes.

These routes are used when transmitting a message to a remote node.
Acquire source routes

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

- The data collector can issue a network discovery command (**ND**) to force all XBees to send a network discovery response. Each network discovery response will be prefaced by a route record.
- Periodic IO sampling can be enabled on remotes to force them to send data at a regular rate. Each IO sample would be prefaced by a route record.
- If the node identifier (**NI**) of the remote XBee is known, the **DN** command can be issued with the **NI** of the remote in the payload. The remote node with a matching NI would send a route record and a **DN** response.
- If the application on remote nodes periodically sends data to the data collector, each transmission forces a route record to occur.

Store source routes

To store source routes for remote nodes:

1. Remote nodes must first send a unicast transmission to the central collector.
2. Upon receipt of a unicast, the XBee emits a Route Record Indicator (0xA1) frame through the serial interface.
3. The information from the route record frame must be interpreted and stored by the application on the microcontroller for later use.

Transmit data

To transmit data using source routing:

1. The microcontroller application must force the XBee to create a source route in its internal source route table by sending a Create Source Route (0x21) frame through the serial interface.
2. After setting up the source route, the application can send data to be wirelessly transmitted (Transmit Request (0x10), Explicit Addressing Command Frame (0x11), or Remote Command Request (0x17) frames) as needed, to the same destination, or any destination in the established source route.
3. If new data must be sent to a destination not included in the last source route, the application must first send a new Create Source Route API frame.

Note If a Create Source Route API frame (0x21) does not precede data frames, data loss may occur.

XBee modules can buffer one source route that includes up to 30 hops (excluding source and destination).

When source routing is used, the 16-bit addresses in the source route are inserted into the RF payload space. This means the payload will be reduced by two bytes per intermediate hop. The user must account for the payload size reduction when using source routing.

Radio firmware

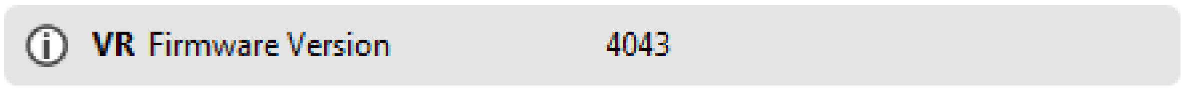
Radio firmware is the program code stored in the radio module's persistent memory that provides the control program for the device. The firmware programmed may determine the protocol of the radio (802.15.4, Zigbee, DigiMesh, or Wi-Fi) if several are compatible, or, in some cases, the role of the module or its operating mode.

Update the firmware of an XBee module locally by connecting the module to your computer or over the air if the module is remotely located. Both methods are supported by XCTU.

Firmware identification

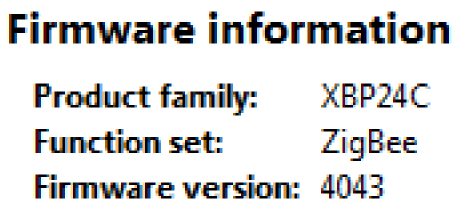
Identify the firmware of an XBee module using three elements:

- **Product family** indicates the XBee type. The product family of the XBee module is printed on the back of the module.
- **Function set** determines the available functionality. For some modules this may include choose transparent or API mode; or whether the device is an end device, router, or coordinator. There are also function selections that allow you to choose firmware for several of Digi's special sensor and adapter modules.
- **Version** is a unique number used to identify the firmware release. The firmware version of an XBee module is reported by the Firmware Version (**VR**) parameter.



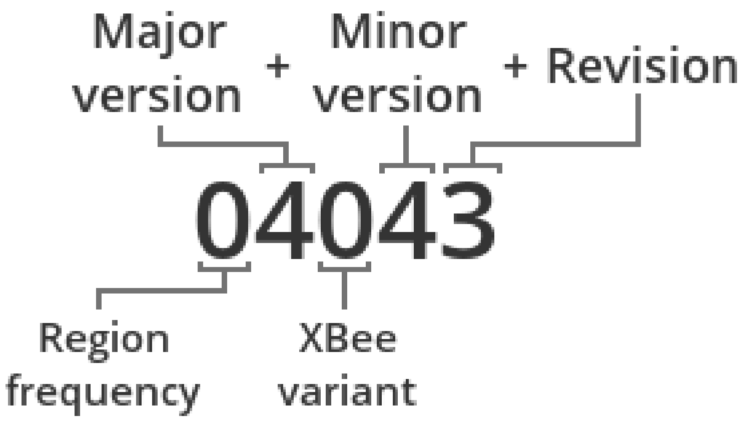
VR Firmware Version 4043

XBee firmware version numbers have five hexadecimal digits using "ABCDE" convention. "A" is an optional digit and if it is not present, it assumes a 0.



Firmware information

Product family:	XBP24C
Function set:	ZigBee
Firmware version:	4043



Update radio firmware

Upgrading the radio firmware of an XBee device can be a common task. For example, you need to update firmware regularly if you want to keep your modules up-to-date and take advantage of improvements and new features implemented in the latest version. You can also change the firmware when you need to use a different function set.

XCTU allows you to upgrade or change the firmware of XBee devices physically attached to your computer or located in remote places over the air using the **Update firmware** tool.

XCTU includes a set of radio firmware files that you can use to update your modules. The Update firmware tool filters these available firmware versions to only list those that are compatible with the selected XBee module.

The complete firmware update process is described in the XCTU User Guide (**Help > Help Contents**). For more information about using the Update firmware tool, see How-to: Update the firmware.

Download new firmware

Digi periodically releases new versions of radio firmware that fix issues, improve functionality, or add new features. Digi also launches new XBee modules in the market that require new radio firmware to be configured with XCTU. These firmware files might not be included with XCTU and need to be downloaded.

XCTU has the ability to download and install the radio firmwares library from the application itself.

By default, XCTU is configured to automatically look for new radio firmware inside Digi's update site when XCTU is started. You can manually launch this process and install firmware previously downloaded or provided by Digi.

1. In XCTU, select **Help** > **Update the Radio Firmware Library**.
   - To look for new firmware inside Digi's update site, select **Remote server**.
   - To add a local XBee firmware file to the XCTU library, select **Local file** and specify the path where the file is located.
2. Click **OK** to start. A dialog displays the status of the download process.

When the process finishes, a new dialog displays the list of downloaded firmware.

Note Downloading the firmware does not automatically update attached modules.

Troubleshooting

If you encounter problems while working on your XBee Zigbee Mesh Kit, try the following troubleshooting tips.

XCTU

Error upon installation of XCTU

XCTU requires Administrator permissions. Check that you have Administrator access on the machine where you are installing XCTU. You may need to request permission to install or run applications as administrator from your network manager.

On Windows systems, a User Account Control dialog may appear when you install XCTU or try to run the XCTU program. You must answer **yes** when prompted to allow the program to make changes to your computer, or XCTU will not work properly.

Discovery process either does not find devices, or XCTU does not list serial ports

Try the following troubleshooting methods:

- **Check cables**: Double check all cables. Make sure the USB cable is firmly and fully attached to both the computer and the XBee Grove Development Board. When attached properly, the association LED on the adapter is lit.
- **Check that the XBee is fully seated in the XBee Grove Development Board**: When the XBee module is properly installed, it should be pushed fully into the board and no air or metal should be visible between the plastic of the adapter socket and the XBee headers. Also, double check that all ten pins on each side of the XBee module made it into a matching hole in the socket.
- **Check the XBee orientation**: The angled "nose" of the XBee should match the lines on the silk screening of the board and point away from the USB socket on the XBee Grove Development Board.
- **Check driver installation**: Drivers are installed the first time the XBee Grove Development Board is plugged in. If this process is not complete or has failed, try the following steps:
  - Remove and re-insert the board into your computer. This may cause driver installation to re-occur.
  - Remove and re-insert the board into another USB port.
  - (Windows) Open Computer management, find the failing device in the Device Manager section and remove it.
  - You can download drivers for all major operating systems from FTDI for manual installation.
- **Check whether the modules are sleeping**: The On/Sleep LED of the XBee Grove Development Board indicates if the module is awake (LED on) or asleep (LED off). When a module is sleeping, it cannot be discovered in XCTU. Press the **Commissioning** button to wake a module for 30 seconds.

XCTU reports errors for KY and DD settings after resetting to factory defaults

This is a known issue with XCTU version 6.1.2 and earlier. When the Invalid settings dialog appears, it is safe to continue writing settings.

- AES Encryption Key (KY) is a setting that must be set by the user when encryption is in use and does not apply with factory settings.
- Device Type Identifier (DD) is a diagnostic parameter which is not used in the operation of the radio and can safely be set to any value.

Basic communication example

The modules are not found in the 'Check the network' step.

Check the Associate LED of ROUTER and END_DEVICE. If it is not blinking, it means that the module is not joined to the network yet. Wait a few seconds and try again.

If the module that is not found is END_DEVICE, it may be asleep. Press the Commissioning button of the board where it is plugged in to wake it up and try again.
The text typed in the end device's console is not sent to the coordinator.

END_DEVICE is configured to cyclically sleep for five seconds and be awake for five seconds. It may have been asleep when you typed in its console, so press the Commissioning button of the XBee Grove Development Board where it is plugged in to wake it up and try again.

To identify the END_DEVICE module, look for the board where the On/Sleep LED is ON for five seconds and OFF for another five seconds.

Wireless data transmission

The application does not find the END_DEVICE module in the network.

This usually occurs when the end device is asleep. Press the Commissioning button of the board where it is attached to wake it up and launch the application again.

To identify the END_DEVICE module, look for the board where the On/Sleep LED is ON for 5 seconds and OFF for another 5 seconds.
Error: Parsing text error message

If you see the "Error parsing the text..." error when sending a message, this indicates that you typed the message incorrectly. Remember to follow this pattern when sending a message:

- Unicast: NODE_IDENTIFIER: message

  For example, to send the message "Hi XBee" to XBEE_C:

  ```
  XBEE_C: Hi XBee
  ```

- Broadcast: ALL: message

  For example, to send the message "Hi XBees" to all nodes in the network:

  ```
  ALL: Hi XBees
  ```

Error: Could not find the module in the network

This message indicates that the module attached to your computer could not send the message to the device whose node identifier is <XXXX>.

Ensure that you typed the node identifier correctly and that it is in the list of devices found by the application. If it is not there, launch the application again.
When I send a message from END_DEVICE, I get the "Error transmitting message..." message.

This happens when the end device is asleep. Wait until it wakes up or press the Commissioning button and send the message again.

To identify the END_DEVICE module, look for the board where the On/Sleep LED is ON for 5 seconds and OFF for another 5 seconds.

The end device receives the messages with some delay.

When the end device is asleep, it cannot receive any message. Its parent stores the message until it wakes up.

Enable sleep mode

ED_PIN does not receive any message.

Ensure that the module is joined to the network. To verify this, check that the Associate LED of ED_PIN is blinking when the DTR button is activated. If not, reset the module and wait a few seconds.

XBee Java library

Warning message: RXTX version mismatch

If you launch an application and see the message "WARNING: RXTX version mismatch", this indicates that the versions of the JAR file and the native library are not the same. You can safely ignore this message.

Invalid operating mode exception message

If you launch an application and you see the "com.digi.xbee.api.exceptions.InvalidOperatingMode" exception, review the following solutions.

- Could not determine operating mode:

```
com.digi.xbee.api.exceptions.InvalidOperatingModeException: Could not determine
operating mode.
       at com.digi.xbee.api.XBeeDevice.open(XBeeDevice.java:211)
       at com.digi.xbee.sendreceivedatasample,MainApp.main(MainApp.java:43)
```

In this case, the library cannot access the module. Check that the PORT constant is correct.

- Unsupported operating mode:

```
com.digi.xbee.api.exceptions.InvalidOperatingModeException: Unsupported
operating mode: AT mode
       at com.digi.xbee.api.XBeeDevice.open(XBeeDevice.java:214)
       at com.digi.xbee.sendreceivedatasample.MainApp.main(MainApp.java:43)
```

This indicates the module is in transparent mode. Change the AP parameter through XCTU to be API enabled [1].

Java lang unsatisfied exception message

If you experience the "java.lang.UnsatisfiedLinkError" exception, there are several possibilities.

- "no rxtxSerial in java.library.path thrown while loading gnu.io.RXTXCommDriver":

```
java.lang.UnsatisfiedLinkError: no rxtxSerial in java.library.path thrown while
loading gnu.io.RXTXCommDriver
       at java.lang.ClassLoader.loadLibrary(ClassLoader.java:1878)
       at java.lang.Runtime.loadLibrary0(Runtime.java:849)
       at java.lang.System.loadLibrary(System.java:1087)
       at gnu.io.CommPortIdentifier.<clinit>(CommPortIdentifier.java:123)
       at com.digi.xbee.api.connection.serial.SerialPortRxTx.open
(SerialPortRxTx.java:161)
       at com.digi.xbee.api.XBeeDevice.open(XBeeDevice.java:189)
       at com.digi.xbee.sendreceivedatasample.MainApp.main(MainApp.java:43)
```

This exception indicates that the RXTX native library is not linked to the rxtx-2.2.jar file. See the second step of the Link the libraries to the project section.

- "Can't load AMD 64-bit .dll on a IA 32-bit platform thrown while loading gnu.io.RXTXCommDriver" (or similar message):

```
java.lang.UnsatisfiedLinkError:
C:\Users\user\workspace\SendReceiveDataSample\libs\native\Windows\win64\rxtxSer
ial.dll: Can't load AMD 64-bit .dll on a IA 32-bit platform thrown while
loading gnu.io.RXTXCommDriver
       Exception in thread "main" java.lang.UnsatisfiedLinkError:
C:\Users\user\workspace\SendReceiveDataSample\libs\native\Windows\win64\rxtxSer
ial.dll: Can't load AMD 64-bit .dll on a IA 32-bit platform
       at java.lang.ClassLoader$NativeLibrary.load(Native Method)
       at java.lang.ClassLoader.loadLibrary1(ClassLoader.java:1957)
       at java.lang.ClassLoader.loadLibrary0(ClassLoader.java:1882)
       at java.lang.ClassLoader.loadLibrary(ClassLoader.java:1872)
       at java.lang.Runtime.loadLibrary0(Runtime.java:849)
       at java.lang.System.loadLibrary(System.java:1087)
       at gnu.io.CommPortIdentifier.<clinit>(CommPortIdentifier.java:123)
       at com.digi.xbee.api.connection.serial.SerialPortRxTx.open
(SerialPortRxTx.java:161)
       at com.digi.xbee.api.XBeeDevice.open(XBeeDevice.java:189)
       at com.digi.xbee.sendreceivedatasample.MainApp.main(MainApp.java:43)
```

This exception indicates that the RXTX native library you have linked is not the correct one. Check to be sure you are not using a 32-bit JVM linked the 64-bit library, or vice versa.
Interface in use exception message

If you experience the "com.digi.xbee.api.exceptions.InterfaceInUseException" exception, it indicates that the port you are trying to open is already in use. Ensure that you do not have any applications running and that the XCTU console of that port is not connected.

```
com.digi.xbee.api.exceptions.InterfaceInUseException: Port COM5 is already in
use by other application(s)
       at com.digi.xbee.api.connection.serial.SerialPortRxTx.open
(SerialPortRxTx.java:189)
       at com.digi.xbee.api.XBeeDevice.open(XBeeDevice.java:189)
       at com.digi.xbee.sendreceivedatasample.MainApp.main(MainApp.java:43)
Caused by: gnu.io.PortInUseException: Unknown Application
       at at gnu.io.CommPortIdentifier.open(CommPortIdentifier.java:467)
       at at com.digi.xbee.api.connection.serial.SerialPortRxTx.open
(SerialPortRxTx.java:167)
       ... 2 more
Error 0x5 at ..\src\termios.c(892): Access is denied.
```

Java lang no class definition exception message

If you experience the "java.lang.NoClassDefFoundError" exception, it indicates that the logger library (**slf4j-api-1.7.7.jar**) is not linked to the project. See the **Link the libraries to the project** section to know which libraries you have to link.

```
Exception in thread "main" java.lang.NoClassDefFoundError:
org/slf4j/LoggerFactory
       at com.digi.xbee.api.connection.serial.AbstractSerialPort.<init>
(AbstractSerialPort.java:170)
       at com.digi.xbee.api.connection.serial.AbstractSerialPort.<init>
(AbstractSerialPort.java:136)
       at com.digi.xbee.api.connection.serial.SerialPortRxTx.<init>
(SerialPortRxTx.java:149)
       at com.digi.xbee.api.connection.serial.SerialPortRxTx.<init>
(SerialPortRxTx.java:124)
```

```
        at com.digi.xbee.api.XBee.createConnectiontionInterface(XBee.java:38)
        at com.digi.xbee.api.AbstractXBeeDevice.<init>(AbstractXBeeDevice.java:164)
        at com.digi.xbee.api.XBeeDevice.<init>(XBeeDevice.java:90)
        at com.digi.xbee.sendreceivedatasample.MainApp.main(MainApp.java:40)
```

SLF4J class path contains multiple bindings message

If you receive the "SLF4J: Class path contains multiple SLF4J bindings" message, it indicates that you linked several logger libraries. Ensure that only the following four libraries are added to your project:

- xbee-java-library-X.Y.Z.jar
- rxtx-2.2.jar
- slf4j-api-x.y.z.jar
- slf4j-nop-x.y.z.jar

XBee Java Library and transparent mode

The XBee Java Library only supports API and API escaped operating modes. You cannot use it with modules in transparent mode.

Receive digital data

No data appears when receiving digital data

Check the following in XBee A (receiver):

1. The value of **ID** is 2015 (the same as the **ID** value of XBee B and XBee C).
2. The value of **CE** is Enabled[1]. The module is a coordinator.

Check the following in XBee B and XBee C (senders):

1. The value of **ID** is 2015 (the same as the **ID** value of XBee A).
2. Channel Verification (**JV**) is enabled.
3. The value of **DH** is 0.
4. The value of **DL** is 0.
5. The value of **D4** is Digital Input [3].
6. The value of **IC** is 10 to changes in DIO4 pin (00010000 binary = 10 hexadecimal).

Note To learn how to configure IC parameter to monitor the pins, see How to obtain data from a sensor.

7. If the data is not received yet, restore the default settings of both XBees and reconfigure them.

Check the following in the Java code:

- The LINE constant that appears on the Java source code is IOLine.DIO4_AD4.

Receive analog data

No data appears when receiving analog data

Check the following in XBee A (receiver):

1. The value of **ID** is 2015 (the same as the **ID** value of XBee B and XBee C).
2. The value of **CE** is Enabled[1] (the module is a coordinator).

Check the following in XBee B and XBee C (senders):

1. The value of **ID** is 2015 (the same as the ID value of XBee A).
2. Channel Verification (**JV**) is enabled
3. The value of **DH** is 0.
4. The value of **DL** is 0.
5. The value of **D3** is ADC [2] if the XBee module is through-hole (THT). If the module is surface-mount (SMT), configure D2 as ADC [2] instead.
6. The value of **IR** is 1388 (5 seconds).

Note To learn how to configure **IR** parameter to monitor the pins, see How to obtain data from a sensor.

7. If the data is not received yet, restore the default settings of both XBee modules and reconfigure them.

Check the following in the Java code:

- The LINES constant that appears on the Java source code contains IOLine.DIO2_AD2 and IOLine.DIO3_AD3.

Send digital actuations

Message: Could not find the module <XXXX> in the network

This message indicates that the module attached to your computer could not establish the connection with the device whose node identifier is the value REMOTE_NODE_ID constant. Check the following:

1. Both modules are in the same network.
2. The Node Identifier (**NI**) of the XBee B (receiver) is ROUTER.

The default **NI** value is a blank space. Make sure to delete the space when you change the value.

In addition, the value of the REMOTE_NODE_ID constant that appears in the Java source code should be ROUTER.

The LED does not blink

Check the following:

1. In the XBee B (RECEIVER), the value of the **D4** setting is Digital Out, High [5].
2. The LINE constant that appears on the Java source code is IOLine.DIO4_AD4.

Range test

Error: There are not remote devices discovered for the selected local device

The local device you have selected has no remote devices. Click the Discover remote devices button and XCTU discovers devices on the local device's network.

There are no remote devices to select

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

Check cables

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

Check that the XBee module is fully seated in the XBee Grove Development Board

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

Check the XBee module orientation

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

Check that the XBee modules are in the same network

Check that the Network ID (**ID**), Preamble ID (**HP**), and Channel Mask (**CM**) settings have the same value for both XBee modules.

Restore default settings

If the XBee modules are properly connected and in the same network, restore default settings and configure them again.
The local RSSI and the number of packets received are always 0

Check cables

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

Check that the XBee module is fully seated in the XBee Grove Development Board

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

Check the XBee module orientation

The angled "nose" of the XBee module should match the lines on the silk screening of the board and point away from the USB socket on the XBee Grove Development board.
Remote RSSI is not included in the chart and the Remote RSSI control is disabled

The **local device** (the one attached to your computer) can be configured to use **API or transparent mode**. The remote device RSSI value can only be read when the local XBee is working in API mode.

To display the remote RSSI value, **reconfigure the local module** to work in API mode.

Parameter	Value	Effect
AP	API enabled [1]	Enables API mode.

Explicit data

After scanning the network, I get the "NO devices found" message.

This message indicates that no devices were found in the network. Ensure you have properly configured all your devices as it is described in the Configure the XBees step.

When I send a message, I get the "Could not find the module '<XXXX>' in the network" message.

This message indicates that the module attached to your computer could not send the message to the device whose node identifier is <XXXX>.

Ensure that you typed the node identifier correctly and that it is in the list of devices found by the application. If it is not there, launch the application again.

Additional resources

Wireless connectivity offers almost unlimited options for making our surroundings smarter, more efficient, and more connected. Now that you have completed the activities in this kit, here are some additional resources to help you explore XBee modules.

Buying considerations

You have become familiar with the XBee modules included in the kit, but Digi makes a large variety of modules with different features and for different functions. So, which module is best suited for your applications? Why are there different types of antennas? Should you use a "PRO" version, or is a regular XBee module enough? In this guide, we look over the different XBee options to help you answer these questions.

Note that the XBee module you select affect the parameters of your application:

- The location of your application affects the operating frequency of the XBee modules.
- To get greater range, you may select an external antenna, a different operating frequency, or even an XBee-PRO.
- Power consumption is an important factor to consider.
- The required network topology also impacts the type of module you need.

To help you select an XBee module based on your requirements, see the XBee Buying Guide.

The following sections review the different options available for XBee radios and how they affect wireless communication.

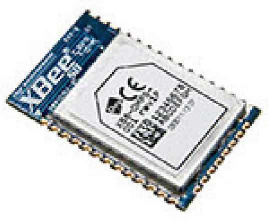
Note Not all options are available for every XBee device.

Hardware footprint

XBee modules come in three hardware footprints: through-hole, surface mount, and micro mount.

- **Through-hole technology (THT)** XBee modules include the 20-pin socket and require holes for mounting the component on the printed circuit board (PCB), although it is common for the carrier board to contain a female socket.
- **Surface-mount technology (SMT)** XBees include 37 pads. They are placed directly on the PCB, which means they do not require holes or sockets for mounting the component.
- **Micro-mount technology (MMT)** XBees include 34 pads. They are placed directly on the PCB, which means they do not require holes or sockets for mounting the component.

XBee Through-hole (THT)


XBee Surface-Mount (SMT)

XBee Micro-Mount (MMT)

Through-hole technology is typically used in prototyping and production. MMT/SMT is recommended for high-volume applications, as the component can be placed automatically by a pick-and-place machine and you save the cost of a socket on each board.

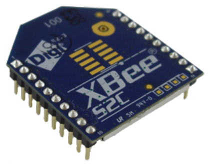
Note Not all XBees are available in all form factors.

XBee antennas

Antennas are devices that focus energy in a particular direction. The attributes of a given antenna affect not only the range of a module but also its price. The following are some potential antenna options available on XBee modules.

Note Not all XBee modules are available in all antenna options:

A **Chip** antenna is mounted directly onto the module and works in conjunction with GND planes on the host PCB.



A **PCB antenna** is formed directly on the module with conductive traces. A PCB antenna performs about the same as a wire antenna and within 5% of the whip antenna.

An **integrated wire antenna** consists of a small wire (about 80 mm) sticking up perpendicular to the PCB. It utilizes a 1/4-wave wire soldered directly to the PCB of the OEM module.

A **whip antenna** is a solid but flexible wire antenna that protrudes about 25 mm above the surface of the XBee PCB. It can be moved around to maximize signal strength, or to stick out of an enclosure. Because it can be moved, it can also be broken off if care is not taken or if the solder connection becomes stressed. It has a range advantage over the chip antenna but only when used outdoors.

Note Folding this antenna parallel with the module will greatly reduce range as the module serves as a ground plane.

A **U.FL antenna** is a tiny connector for your own external antenna. Typical connection is either a dipole antenna with U.FL connection, or a U.FL to RP-SMA adapter cable. This is a good option if your XBee is in a box and you want your antenna outside the box.

A **RP-SMA antenna** (reverse-polarity SMA) is a bigger connector for your external antenna. XBee radios are equipped with an RP-SMA female plug, and the antenna is an RP-SMA male jack. This is another good option if your XBee is in a box and you want your antenna outside the box.

XBee vs. XBee-PRO

Both XBee and XBee-PRO modules are small, high-performance, low-cost, wireless data radios. They are pin-compatible with one another and you can mix and match them on the same network. However, there are a few differences between them:

- XBee-PRO TH modules are slightly longer than regular XBees (except for TH XBee3 modules).
- XBee-PRO modules typically use more power.
- The XBee-PRO has a longer range than the XBee.
- The XBee-PRO typically has a higher MSRP.

XBee

XBee-PRO

Frequency

XBee modules are available in a wide variety of RF frequencies used around the world. XBee frequency affects range, and it also affects the location where you can deploy your application due to regulation differences between countries. Modules with different frequencies **cannot** be mixed on the same network. Developers can support frequencies including:

- 2.4 GHz
- 902 - 928 MHz
- 865 - 868 MHz

To determine the frequency that is best for your application, you need to answer two questions:

1. **Where is your application going to be deployed?**

   Since some frequencies are not allowed for unlicensed use in certain countries, it is important to consider the location of your application when you select the frequency of your XBees.

   Note XBee devices have been certified for use in certain countries. See the specific product manual for the latest details.

- 900 MHz radio frequency band is for unlicensed use only in North America and Australia.
- 868 MHz radio frequency band is for unlicensed use only in Europe.
- 865 MHz radio frequency band is for unlicensed use only in India.
- 2.4 GHz is an unlicensed radio frequency band used worldwide.

2. **What is the maximum range your application needs to communicate?**

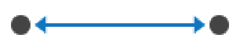
   900 MHz, 868 MHz, and 865 MHz XBee modules offer much greater range than 2.4 GHz XBee modules.

Radio communication protocols

XBee modules support multiple wireless protocols which are suitable for many different network topologies. Open standards include Zigbee, 802.15.4, and Wi-Fi. Digi has also developed proprietary protocols such as Multipoint and DigiMesh. The following list includes the supported protocols:

IEEE 802.15.4

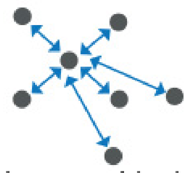
IEEE 802.15.4 is a standard which specifies the physical layer and media access control for low-rate wireless personal area networks. It is the basis for the Zigbee, ISA100.11a, WirelessHART, and MiWi specifications, each of which further extends the standard by developing the upper layers which are not defined in IEEE 802.15.4. The modules included in this kit are IEEE 802.15.4. The standard is designed specifically for energy efficient communications in a point-to-point or a point-to-multipoint configuration and includes sleeping and security.



Point-to-point

Use:

- Single point communications.
- Fast connections between two devices.



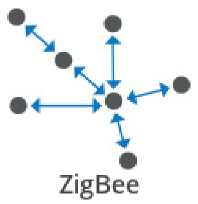
Point-to-multipoint

Use:

- Non-expandible networks that need low power or intermittent functioning.

Zigbee / Zigbee SE (Smart Energy)

Zigbee is a specification for a suite of high-level communication protocols. Its main purpose is to create a meshed network topology (hierarchy) to allow a number of devices to communicate among them.

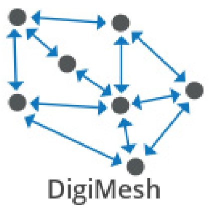


Use:

- Large systems that need to expand without a loss of function.
- Systems that need extended communications.
- Systems using non-directional communications patterns.
- Systems with intermittent function of the individual modules due to power loss or cyclical functioning.
- Large-scale networks with low power functioning on end devices.
- Linked embedded devices or devices that move.
- Systems that require interoperability between devices made by different vendors.

DigiMesh (Digi proprietary)

DigiMesh is a proprietary peer-to-peer wireless networking protocol developed by Digi International Inc. DigiMesh forms a meshed network. The protocol allows for time-synchronized sleeping nodes/routers and low-power battery powered operation. The protocol is currently supported by several 900 MHz, 868 MHz, 865 MHz, and 2.4 GHz Digi radio modules.

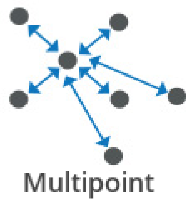


Use:

- Systems that require the ability to sleep on all nodes.
- Systems that require simplified network setup and expansion.
- More robust mesh networks (no parent/child dependencies).
- Systems that require longer range options for each hop.
- Systems with larger frame payloads.
- Environments where increased reliability is important due to routers that come and go due to interference or damage.

Multipoint (Digi proprietary)

XBee multipoint RF modules are ideal for applications requiring low latency and predictable communication timing. They provide quick, robust communication in point-to-point, peer-to-peer, and multipoint/star configurations. Multipoint modules can be deployed as a pure cable replacement for simple serial communication or as part of a more complex hub-and-spoke network of sensors. Many DigiMesh XBee devices also support Multipoint as a transmit option.



Use:

- Multipoint networks with longer range options.
- Systems that do not require fast communication.

IEEE 802.11 (Wi-Fi)

XBee Wi-Fi RF modules provide wireless connectivity to end-point devices in 802.11 bgn networks. Using the 802.11 feature set, these modules are interoperable with other 802.11 bgn devices, including devices from other vendors.

Use:

- Cloud-connected Wi-Fi products.
- Ideal for industrial applications that require fast time to market.
- Easily connect to a smartphone or tablet for configuration or data transfer.

Cellular

Cellular technology is ideal for wireless connectivity to end-point devices in remote locations or where signal penetration is an issue. Easily send data to AWS/MQTT or your own server application using TCP, UDP and SSL/TLS communication protocols.

Use:

- Cloud-connected cellular products.
- Ideal for remote applications that require fast time to market.
- Easily able to connect to AWS services.

Note Not all XBee devices can run all communication protocols. The combination of XBee hardware and radio firmware determines the protocol that an XBee device can execute. Refer to the Digi XBee Family Features Comparison for more information about the available XBee RF modules and the protocols they support.

Where to buy XBee devices

We are committed to providing our customers with local sales and support across the globe. With our reach extending to more than 70 countries worldwide, we work with a number of channel partners in local countries to provide you with excellent sales and support.

Advantages of working with our network of international channel partners include:

- Local language contacts within the organization.
- In-country and local language technical support and customer service.
- In-country RMA support.
- In-country product delivery.
- Understanding of local businesses and industry segments.

Contact us online or by phone at 1-952-912-3444 for more information about which channel partner is best suited to your individual needs.

Find products from Digi and Digi distributors

Digi products are available from many sources worldwide.

- You can find Digi products through distributors. Find a distributor now.
- You can also find Digi products in our official Digi online store:
  - Americas online store (U.S., Canada, and Latin America)
  - EMEA online store (Europe, Middle East, and Africa)
  - Japan online store
  - U.S. Government Sales

Find Digi products through resellers

You can purchase XBees and other Digi networking products from online retailers:

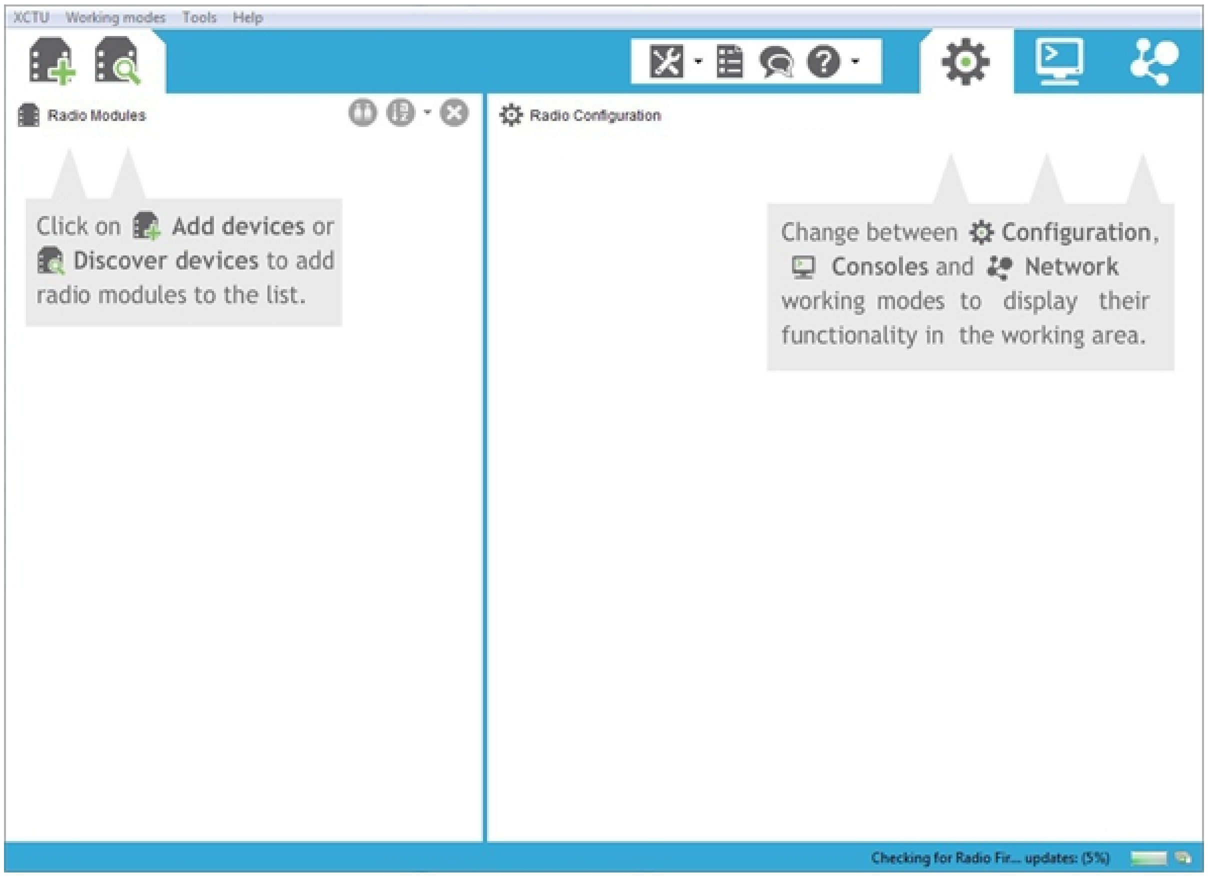
- Adafruit
- Fry's
- Maker Shed
- Microcenter
- Parallax
- RobotShop
- Seeed Studio
- Solarbotics
- Sparkfun
- TrossenRobotics

XCTU walkthrough

This walkthrough describes the layout and basic concepts of the XCTU tool.

XCTU overview

XCTU is divided into five main sections: the menu bar, main toolbar, devices list, working area, and status bar.



Click on Add devices or Discover devices to add radio modules to the list.

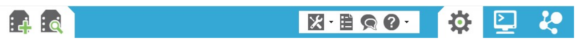
Change between Configuration, Consoles and Network working modes to display their functionality in the working area.

Menu bar

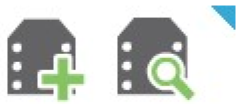
The menu bar is located at the top of the application. You can use the menu bar to access all XCTU features, tools, and working modes.

Main toolbar

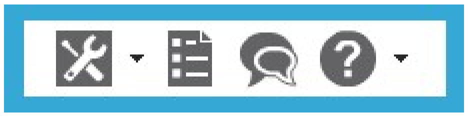
The main toolbar is located at the top of the application and is divided into three sections.



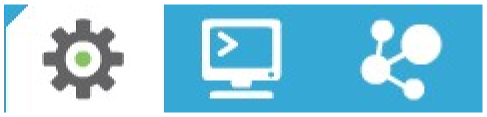
- The first section contains two icons used to add radio modules to the radio modules list. See Add radio modules to XCTU.



- The second section contains the static XCTU functionality that does not require a radio module. This section includes the XCTU tools, the XCTU configuration, the feedback form, and the help and updates functions. See XCTU tools and Configure XCTU.



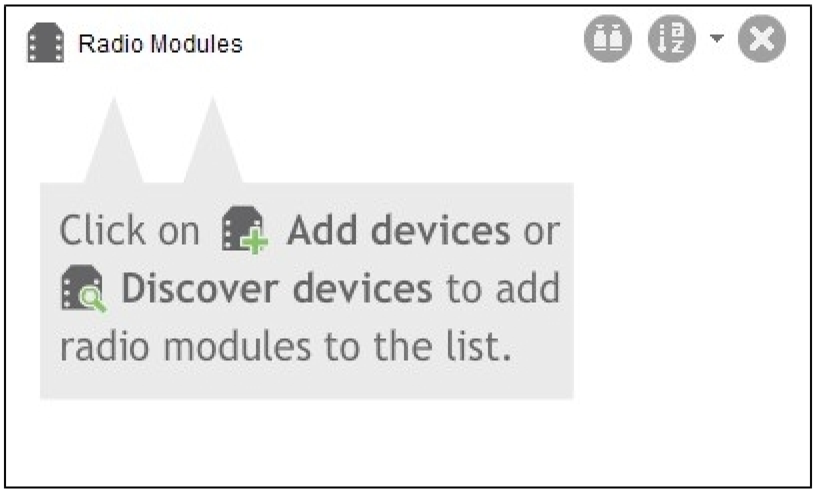
- The third section contains tabs corresponding to the three XCTU working modes. To use this functionality, you must have added one or more radio modules to the list. See XCTU working modes.



Devices list

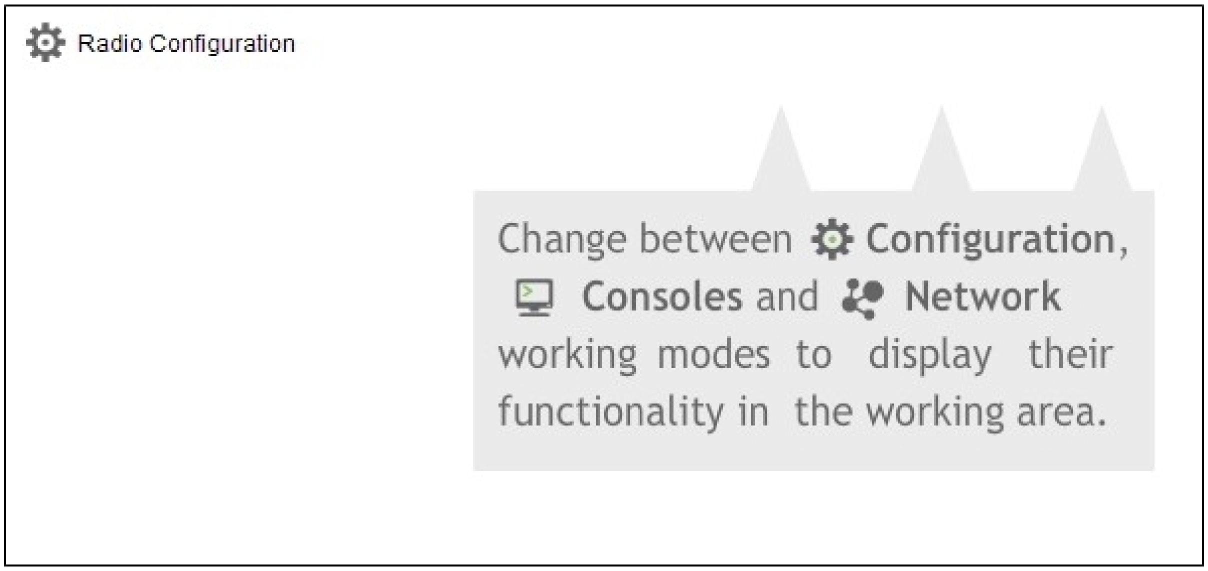
The radio modules list, or devices list, is located on the left side of the tool and displays the radio modules that are connected to your computer. If you know the serial port configuration of a radio module, you can add it to the list directly. You can also use the discovery feature of XCTU to find radio modules connected to your PC and add them to the list. See Add radio modules to XCTU.

Depending on the protocol of the local radio modules added, you can also add remote radio modules to the list using the module's search feature.


Radio Modules
Click on Add devices or Discover devices to add radio modules to the list.

Working area

The working area is the largest section and is located at the right side of the application. The contents of the working area depend on the working mode selected in the toolbar. To interact with the controls displayed in the working area, you must have added one or more radio modules to the list and one of the modules must be selected.



Change between Configuration, Consoles and Network working modes to display their functionality in the working area.

Status bar

The status bar is located at the bottom of the application and displays the status of specific tasks, such as the firmware download process.

Application working modes

A working mode represents a layout which displays operations you can perform with a radio module. Typically, the working mode functionality appears in the working area. The tool has 4 working modes:

- **Configuration mode**: Allows you to configure the selected radio module from the list.
- **Consoles mode**: Allows you to interact or communicate with the selected radio module.
- **Network mode**: Allows you to discover and see the network topology of 802.15.4, Zigbee and DigiMesh protocols.
- **Remote Manager mode**: Allows you to learn about the Digi Remote Manager platform, create an account, and access your personal Digi Remote Manager page.

You can only select one working mode at a time. The Configuration mode is the default selection when you start XCTU.

Add a module

To work and interact with a radio module, you must plug it into a USB adapter and plug that adapter into one of your computer's USB ports. Once you have connected the module, you must add it to the list of devices. There are two different methods:

1. If you know the serial configuration of your radio module, click the **Add radio module** button to add it directly.

When the dialog opens, you must select the serial port the radio module is connected to and configure the serial settings of the port.

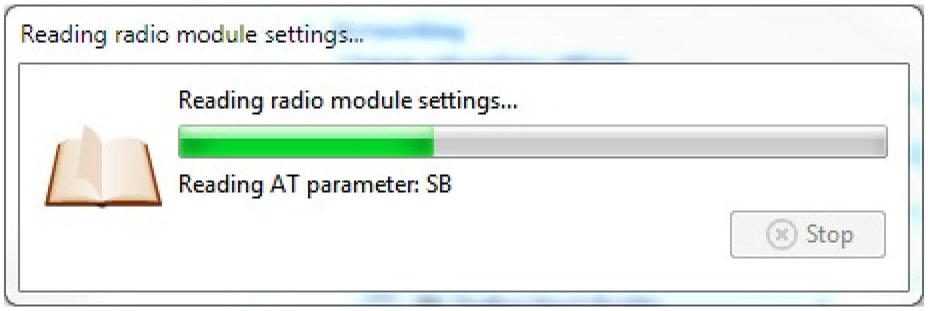
2. If you don't kno the serial configuration of your radio module, or which port is connected to, or if you want to add more than one module, click the the **Discover radio modules** button to use the discovery utility.

In the Discover radio devices dialog, select the serial port in which you want to look for radio modules. If you do not know the serial port where your module is attached, select all ports. Click **Next**, and then click **Finish**.

As radio modules are found, they appear in the Discovering radio modules… dialog box. Once the discovery process has finished, click **Add selected devices**.

Read settings

If you are in the Configuration working mode, when you select a radio module from the list of devices XCTU displays the settings of that module in the working area. XCTU automatically reads the values and completes all the fields.

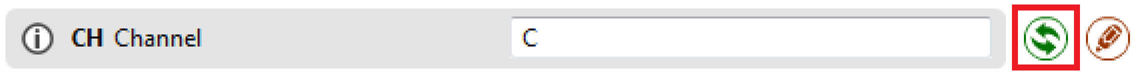

Reading radio module settings...

Reading radio module settings...

Reading AT parameter: SB

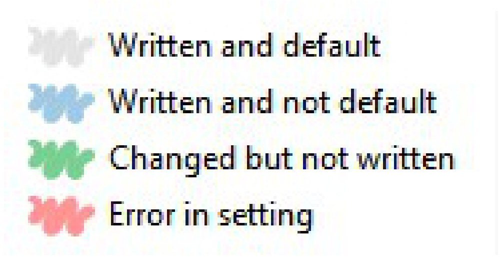
At any time, you can read the settings of the selected radio module. To do so, click the **Read module settings** button and XCTU reads the firmware settings and refreshes the values.

The previous button reads all the settings, but if you want to read only a specific setting you can click the **Refresh** button that appears on the right side of the setting control.


CH Channel C

Change settings

When you change the value of a setting, the background color of the control changes depending on the status of its value. The color legend appears next to the firmware information panel and reads the following:


⬜ Written and default
🟦 Written and not default
🟩 Changed but not written
🟥 Error in setting

- **Gray**: The value of the setting is written in the radio module and matches the default value.
- **Blue**: The value of the setting is written in the radio module but is different than the default value.
- **Green**: The value of the setting has changed but it has not been written in the radio module yet.
- **Red**: The value of the setting is not valid.

Whenever you change the value of a setting, you must save the changes in the module.

Save settings

If you have changed the value of any firmware setting, click the **Write module settings** button to write the new values to the radio module.

As with the reading process, the previous button writes all the settings that have been modified. If you want to write only a specific setting, click the **Write** button that appears on the right side of the setting control.

Real projects with XBee modules

Explore the links below for real-world projects made with XBee technology.

Community

XBee Projects

The largest collection of XBee projects on the web.

Digi XBee examples and guides

Learn more about wirelessly connecting XBees to sensors, outputs, motors, lights, and the Internet.

Industrial solutions

Wireless Tank Monitoring with 1844myfuels
Wireless ultrasonic sensors connected to XBee modules enable up-to-the-minute monitoring of farm silo levels.

Devergy Expands Solar Power Possibilities in Africa
Devergy uses XBee technology for the communication network, where hundreds of nodes are connected with XBee modules—making the solar micro-grids smart, cost effective and manageable.

Tracking Hand Washing Decreases the Spread of Infection at Hospitals
Hand washing is one of the most important daily routines to avoid the spreading of bacteria and disease, especially in hospitals and healthcare centers.

XBee helps Libelium monitor harsh environments
Embedded XBee and XBee-PRO modules enable low-cost, low-power remote monitoring of isolated and difficult-to-access sensors.

For more industrial solutions, visit www.digi.com/industries.

Related products

See the following products from Digi International.

XBee Gateway

The low-cost XBee-to-IP solution enables remote connectivity, configuration, and management of XBee networks with Digi Remote Manager. All XBee data sent to the gateway is automatically available to online applications via Digi Remote Manager. Additionally, this gateway can run custom Python applications that communicate and manage your XBee network.

XBee RF Modems

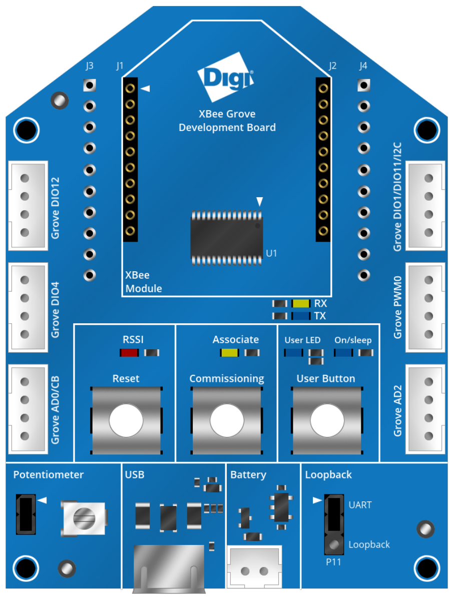
XBee RF Modems are small, low-power devices using XBee RF modules to communicate with systems using RS-232, RS-485, and USB interfaces. You can easily make existing wired systems wireless with this out-of-box solution. XBee RF modems are ideal for extended-range applications with a high data throughput.

XBee Grove Development Board

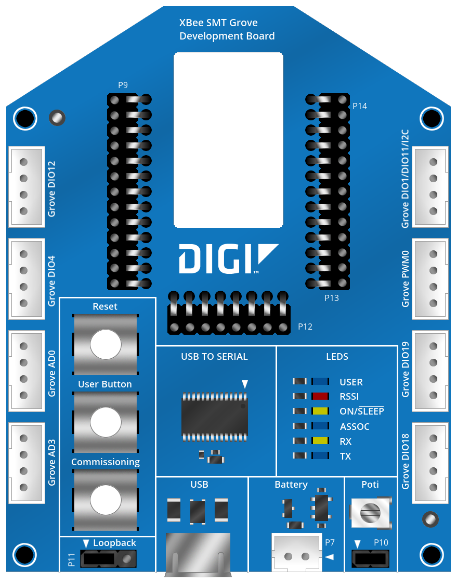
The XBee Grove Development Board is a simple-to-use base unit. You can use it to evaluate XBee modules, as it connects any XBee/XBee-PRO module to a PC or microcontroller. One of the main features of the board is that it has several Grove connectors where you can plug in a Grove Module. You can learn more about the Grove module on the Seeed Studio wiki.

The THT and SMT are the two variants of the board.

XBee THT Grove Development Board



XBee SMT Grove Development Board



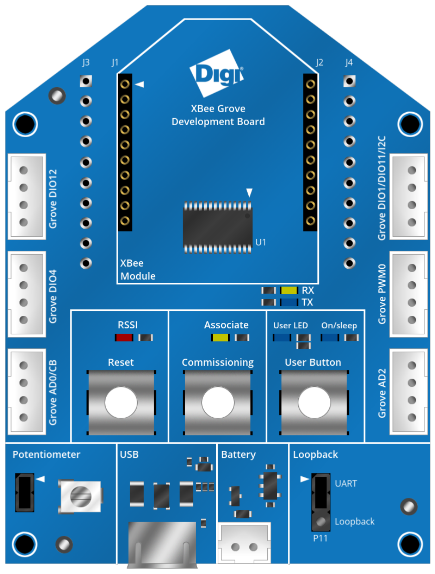
Overview

This section provides an overview of the XBee Grove Development Board.

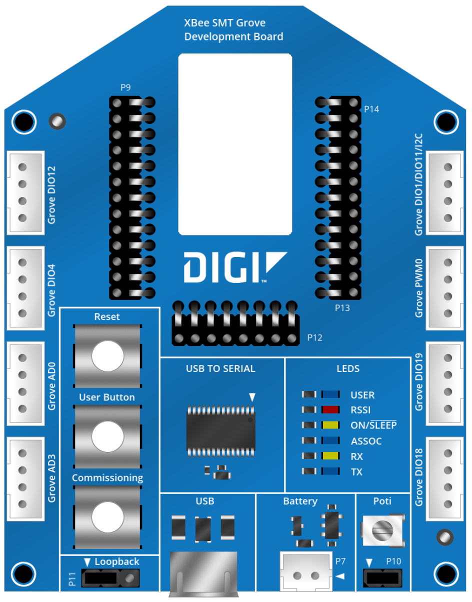
Development board variants

The THT and SMT are the two variants of the board.

XBee THT Grove Development Board



XBee SMT Grove Development Board



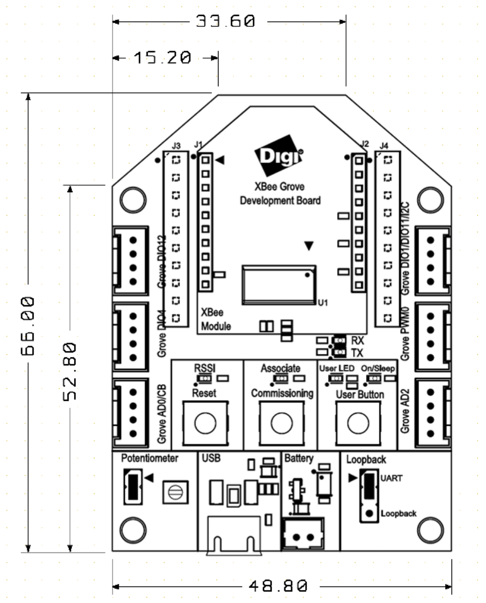
Mechanical

There are two variants of the XBee Grove Development Board:

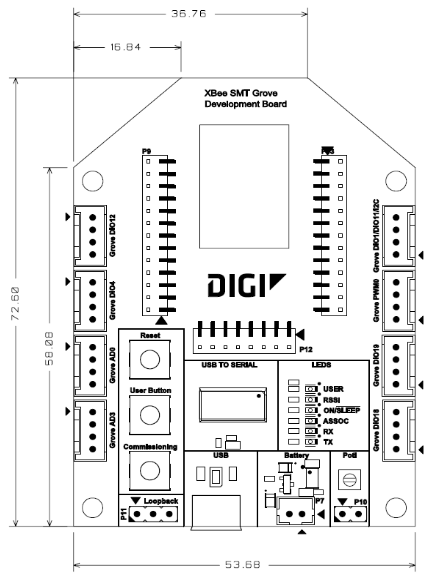
- THT variant is 48.8 mm x 66 mm
- SMT variant is 53.68 mm x 72.60 mm with a shape similar to a regular XBee module.

The board provides four 3.2 mm assembly drills.

XBee THT Grove Development Board variant



XBee SMT Grove Development Board variant



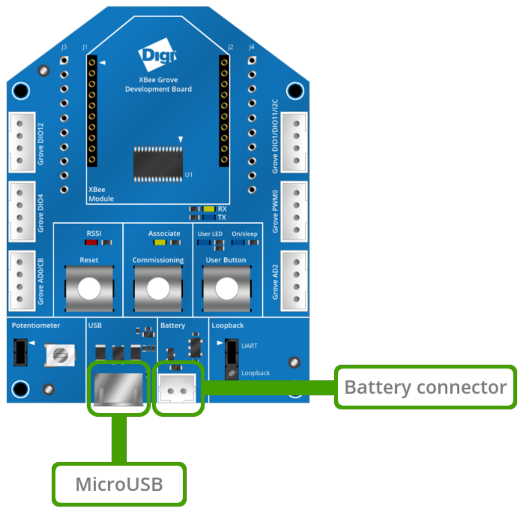
Power supply

You can power the XBee Grove Development Board from the 5 V supply available on the USB connector or from an external battery connected to a 2-pin, 2 mm pitch, PH-type connector from JST. When you power the board from both supplies, it uses the USB.

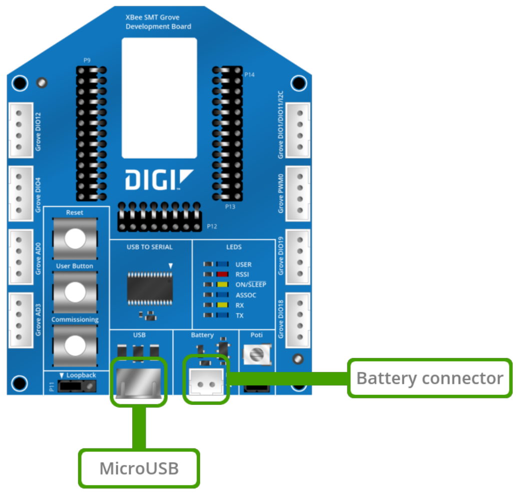
The board has a 3.3 V regulator that generates 500 mA supply.

Note The power supply battery connector is not mounted in the board.

XBee THT Grove Development Board power supply



XBee SMT Grove Development Board power supply



Power supply battery connector

The following table shows the pinout of the battery connector:

Battery connector	Signal	Comments
2	GND	
1	VBAT	Battery supply input

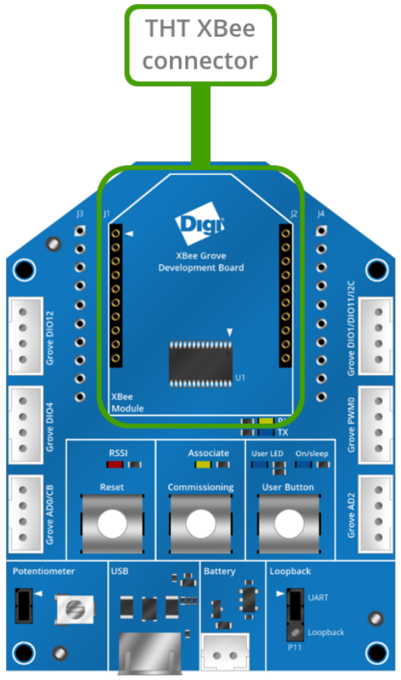
Note The power supply battery connector is not mounted in the board.

XBee connector

The XBee THT Grove Development Board provides two 10-pin, THT, 2 mm pitch sockets to connect a THT XBee module. It is compatible with the XBee/XBee-PRO and the programmable XBee.

XBee THT Grove Development Board XBee connector

The board provides footprints for two 10-pin, THT, 2.54 mm pitch connectors. You can use these footprints to solder a pin header on the top or bottom to access the XBee signals or to connect the XBee Grove Development Board to a bread board.

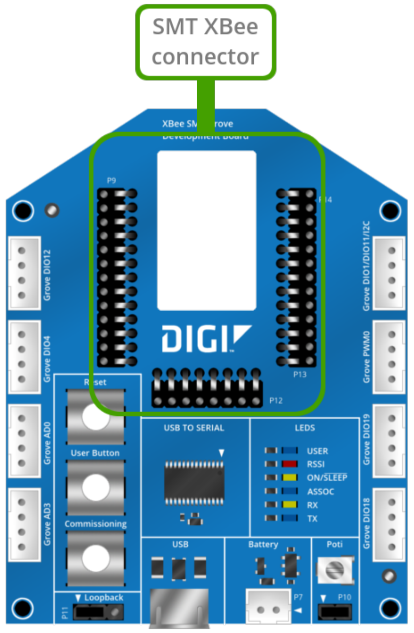


Left			Right		
Pin	**Signal**	**Comments**	**Pin**	**Signal**	**Comments**
1	3.3V	XBee supply	1	DIO4	To GROVE_DIO4 and user LED/button
2	XBEE_TX	To serial to USB device	2	XBEE_CTS_N	To serial to USB device

Left			Right		
3	XBEE_RX	To serial to USB device	3	DIO9	To On/Sleep LED
4	DIO12	To GROVE_DIO12	4	VREF	
5	RESET_N	To reset button	5	ASSOC_LED	To association LED
6	RSSI/PWM0	To RSSI LED and GROVE_PWM	6	XBEE_RTS_N	To serial to USB device
7	DIO11/I2C_SDA	To GROVE_I2C	7	AD3	To potentiometer
8	XBEE_PIN8	Connected to breadboard header	8	AD2	To GROVE_AD2
9	XBEE_DTR_N	To serial to USB device	9	DIO1/ISC_SCL	To GROVE_I2C
10	GND		10	AD0/CB	To commissioning button and GROVE_AD0

XBee SMT Grove Development Board XBee connector

The XBee SMT Grove Development Board provides three spring sockets. A spring header is a custom Digi header that provides a reliable connection to SMT XBee modules without soldering the module in place.



Left			Bottom			Right		
Pin	**Signal**	**Comments**	**Pin**	**Signal**	**Comments**	**Pin**	**Signal**	**Comments**
1	GND		1	DIO18	To GROVE_ DIO18	1		
2	3.3V	XBee supply	2			2	AD0/CB	To commissioning button and GROVE_ AD0

Left			Bottom			Right		
3	XBEE_TX	To serial to USB device	3			3	DIO1/I2C_ SCL	To GROVE_I2C
4	XBEE_RX	To serial to USB device	4			4	AD2	To potentiometer
5	DIO12	To GROVE_DIO12	5			5	AD3	To GROVE_AD3
6	RESET_N	To reset button	6			6	XBEE_RTS_N	To serial to USB device
7	RSSI/PWM0	To RSSI LED and GROVE_ PWM0	7			7	ASSOC_LED	To association LED
8	DIO11/I2C_ SDA	To GROVE_I2C	8			8	VREF	
9	-		9			9	DIO9	To On/Sleep LED
10	XBEE_DTR_N	To serial to USB device	10			10	XBEE_CTS_N	To serial to USB device
11	GND		11			11	DIO4	To GROVE_DIO4 and user LED/button
12	DIO19	To GROVE_DIO19	12			12		
13	GND		13			13		

USB

The XBee Grove Development Board includes a micro USB connector and an FT232RL USB to RS-232 converter to communicate with the serial port of the XBee.

A green LED and a yellow LED show the status of the TX and RX lines.

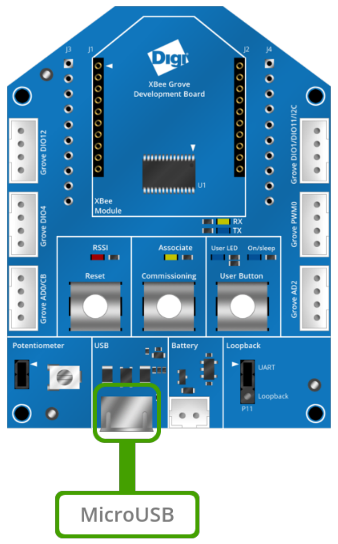
The hardware flow control signals of the XBee (XBee_RTS and XBee_CTS) connect to the FT232RL device. Two serial or resistors disconnect the flow control of the chip if this functionality is not needed.

The XBEE_DTR_N signal is also connected to the FT232 chip. XCTU uses this signal to enter in the boot loader and recover the module from incorrect firmware. A configurable OR resistor disconnects this signal if the functionality is not needed.

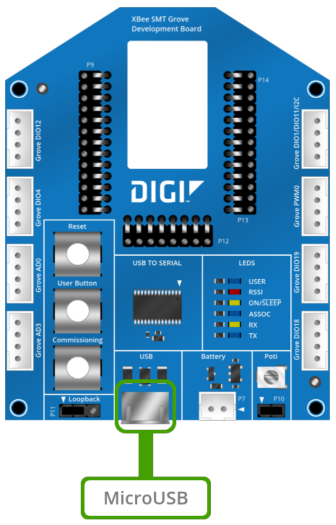
A three-pin jumper configures the serial port in a loopback mode, connecting the RX and TX lines together. When you close positions 1 and 2, the serial port is configured in normal mode and the serial port of the XBee is connected to the micro USB connector. If you close positions 2 and 3, the serial port works in loopback mode and the data transmitted by the XBee connects to the RX pin.

The USB connector also powers the board through the VBUS line.

XBee THT Grove Development Board USB

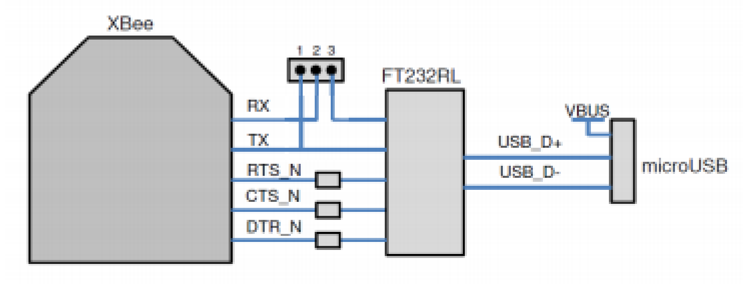


XBee SMT Grove Development Board USB



USB VBUS line

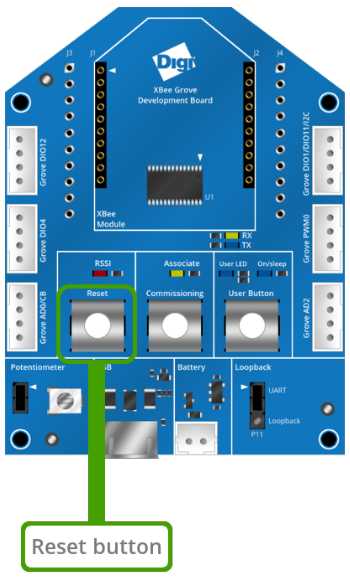
The following graphic illustrates how the USB powers the board through the VBUS line.



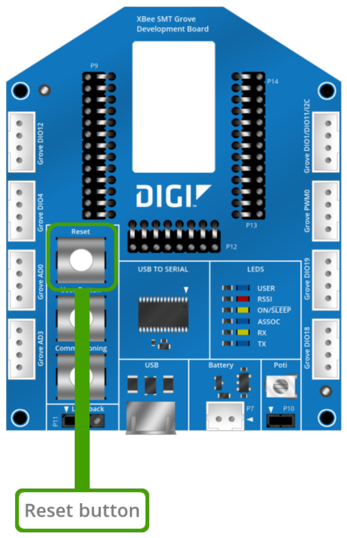
Reset button

The XBee Grove Development Board has a reset button to reboot the XBee module.

XBee THT Grove Development Board Reset button



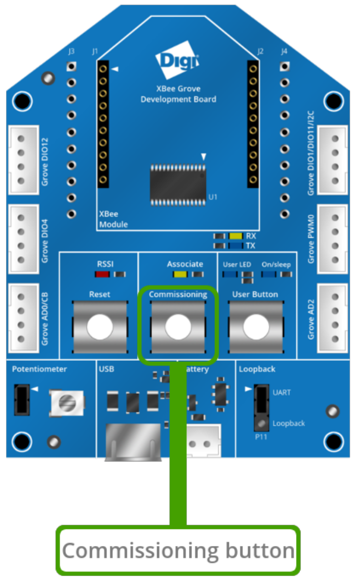
XBee SMT Grove Development Board Reset button



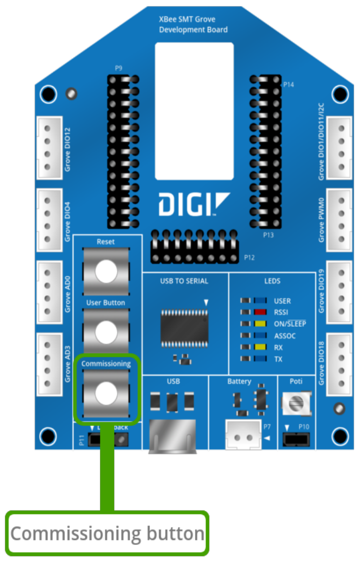
Commissioning button

The XBee Grove Development Board has a push button connected to the commissioning pin of the XBee module. The commissioning pin of the XBee is also connected to the Grove AD0 connector. You can use the commissioning push button in Zigbee or DigiMesh to help deploy devices in a network.

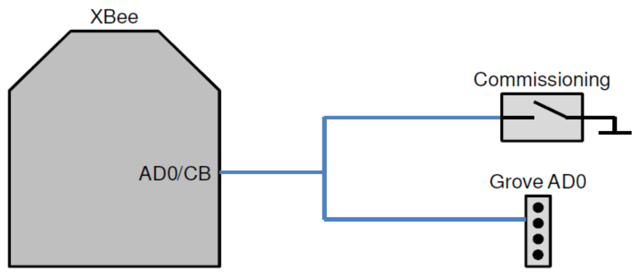
XBee THT Grove Development Board Commissioning button



XBee SMT Grove Development Board Commissioning button



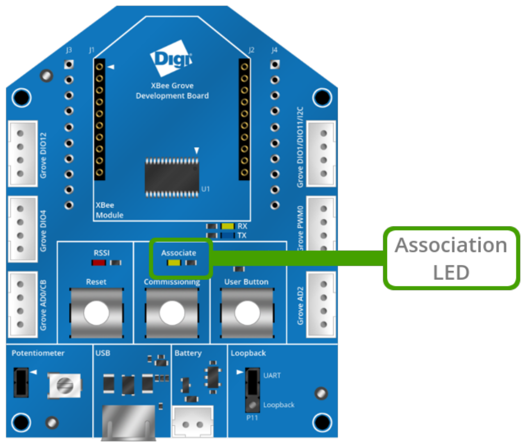
Commissioning pin and Grove AD0 connection



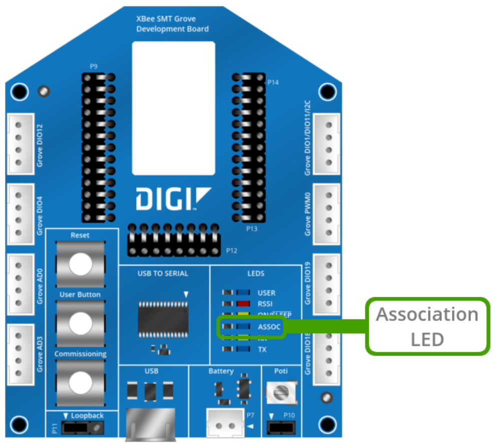
Association led

The XBee Grove Development Board provides an LED connected to the association pin of the XBee module.

XBee THT Grove Development Board Association LED



XBee SMT Grove Development Board Association LED

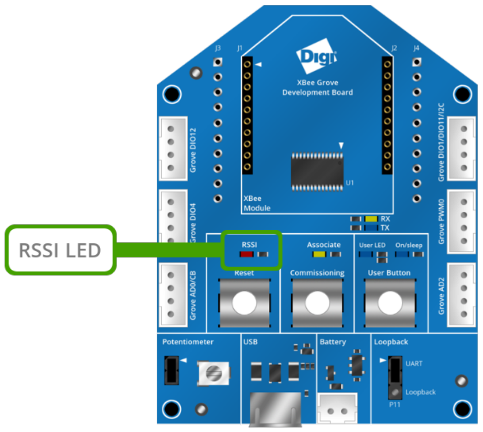


RSSI led

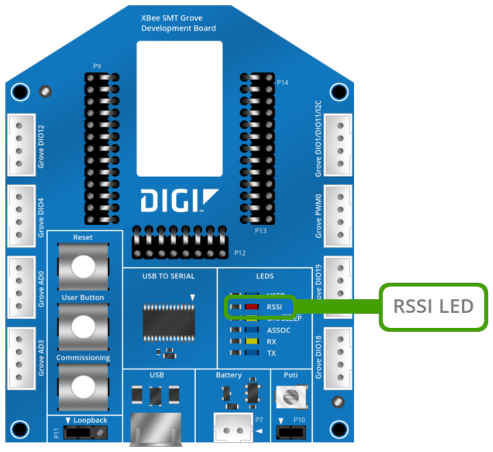
The XBee Grove Development Board provides an LED connected to the RSSI/PWM0 pin of the XBee module. The RSSI/PWM signal is also connected to the PWM Grove connector.

If the PWM0 pin (**P0**) is configured as RSSI, the brightness of this LED displays the signal strength of the last packet received.

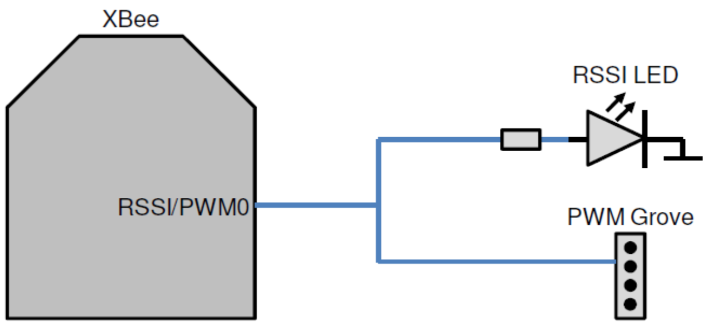
XBee THT Grove Development Board RSSI LED



XBee SMT Grove Development Board RSSI LED



PWM0 RSSI configuration

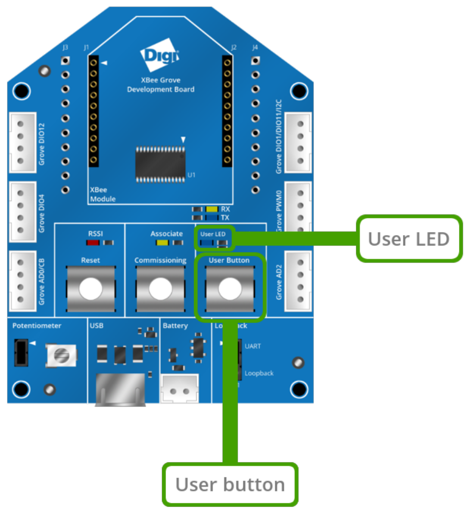


User LED and User button

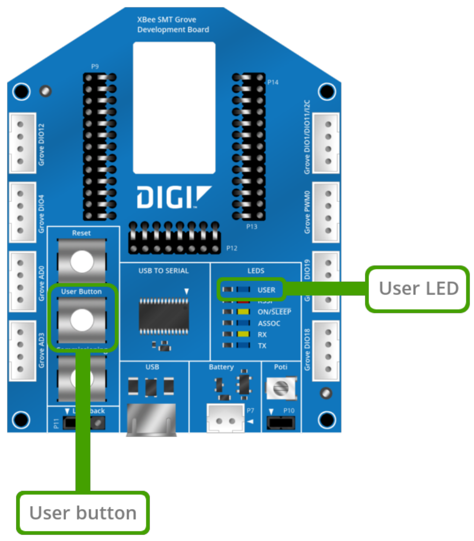
The XBee Grove Development Board provides a user LED and a user button. Both share the same XBee I/O pin, DIO4.

⚠ Although the user LED and user button share the same pin, you can use only one at a time.

XBee THT Grove Development Board User LED and User button

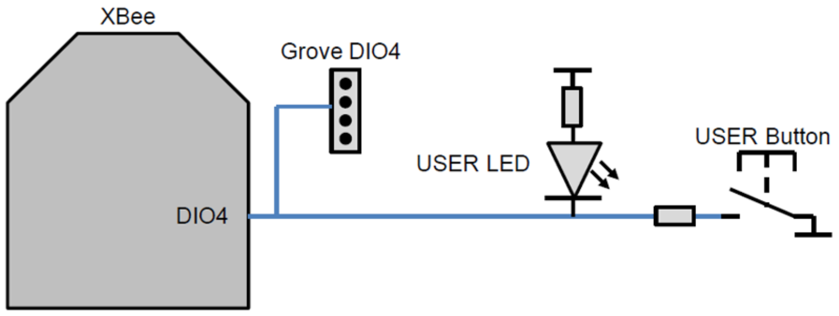


XBee SMT Grove Development Board User LED and User button



User LED and User Button connection to DIO4

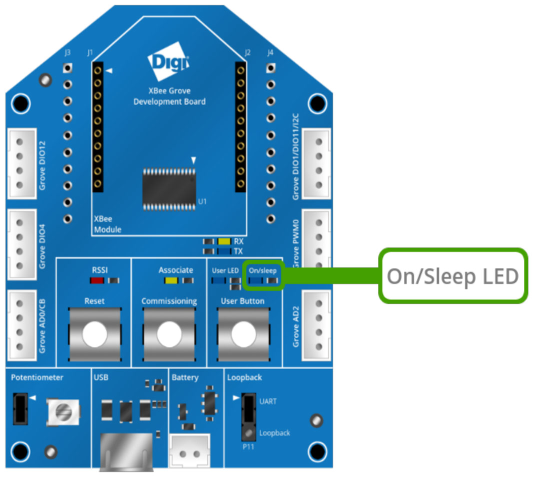
The following graphic illustrates the connection between the User LED and User button to the I/O pin, DIO4.


XBee
Grove DIO4
USER LED
USER Button
DIO4

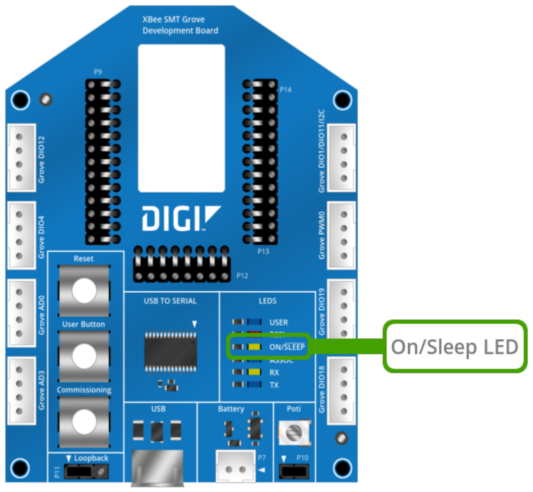
On/sleep LED

The XBee Grove Development Board provides an LED connected to the On/Sleep pin (DIO9). This LED is on when the XBee module is awake, and off when it is asleep.

XBee THT Grove Development Board On/Sleep LED

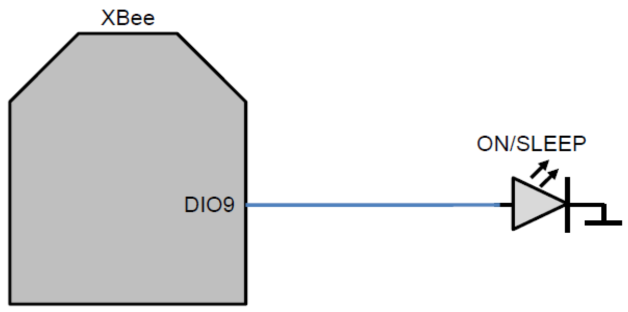


XBee SMT Grove Development Board On/Sleep LED



On/sleep LED connection to DIO9

The following graphic illustrates the connection between the on/sleep LED and the On/sleep pin, DIO9.



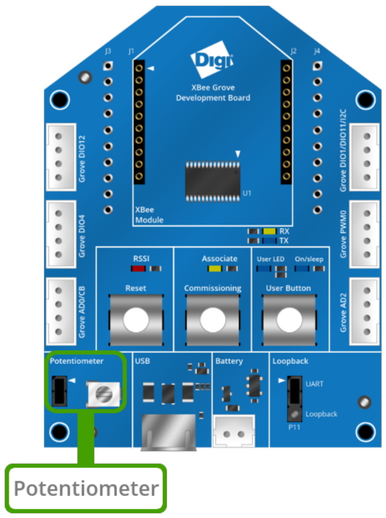
Potentiometer

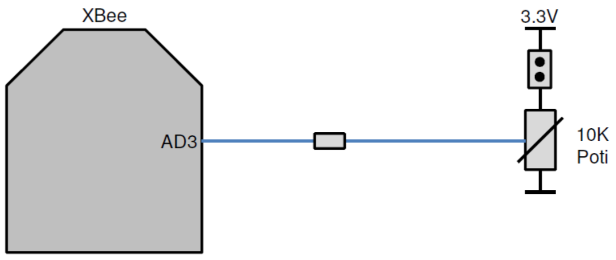
The XBee Grove Development Board provides a 10K potentiometer to generate analog signal between 3.3V and 0V.

You can use the jumper to disconnect the 3.3V supply from the potentiometer to save power when not in use.

XBee THT Grove Development Board Potentiometer

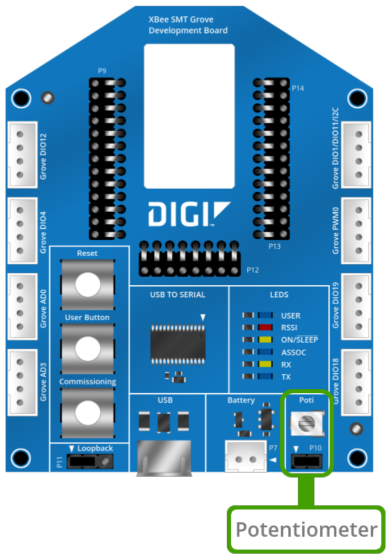
The output of the potentiometer is connected to the AD3 pin (D3) of the XBee in the THT board.

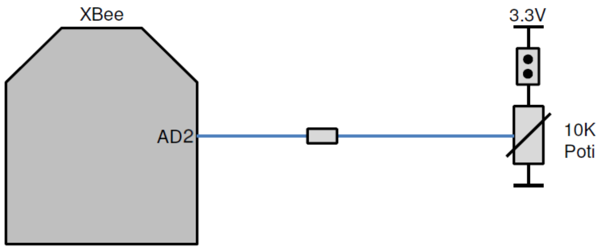




XBee SMT Grove Development Board Potentiometer

The output of the potentiometer is connected to AD2 pin (D2) of the XBee in the SMT board.

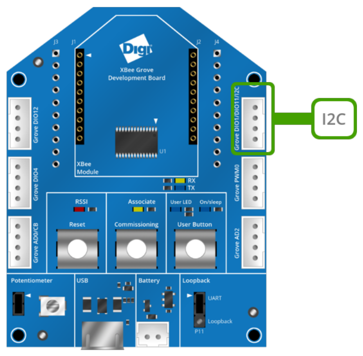




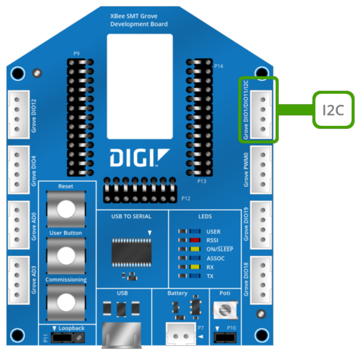
I2C

The XBee Grove Development Board provides an I2C bus that you can use with XBee programmable modules.

XBee THT Grove Development Board I2C bus

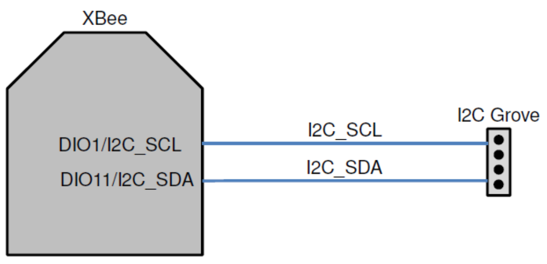


XBee SMT Grove Development Board I2C bus



XBee/XBee-PRO connection to Grove sensor

Regular XBee/XBee-PRO modules do not provide an I2C bus, but you can connect a digital Grove sensor.



Grove I2C connector pinout

The following table shows the pinout of the Grove I2C connector:

Grove I2C	Signal
1	DIO1/I2C_SCL
2	DIO11/I2C_SDA
3	3.3V
4	GND

Grove Connectors

The XBee Grove Development Board provides several Grove connectors connected to the XBee pins:

- THT boards include six Grove connectors:
  - Two connectors to digital I/O pins
  - Two connectors to two digital/analog I/O pins
  - One connector to the RSSI/PWM0 pin
  - One connector to the I2C bus of the microcontroller placed in the socket (programmable XBee)
- SMT boards include eight Grove connectors:
  - Four connectors to digital I/O pins
  - Two connectors to two digital/analog I/O pins
  - One connector to the RSSI/PWM0 pin
  - One connector to the I2C bus of the microcontroller placed in the socket (programmable XBee)

For more information about Grove sensors and actuators for use with these connectors see the Seed Studio wiki.

THT board Grove connectors pinout

The following tables show the pinout for the THT board Grove connectors:

Grove DIO12	Signal	Comments
1	DIO12	
2	-	
3	3.3V	
4	GND	

Grove DIO4	Signal	Comments
1	DIO4	Signal connected to the user LED/button
2	-	
3	3.3V	
4	GND	

Grove AD0	Signal	Comments
1	AD0/CB	Signal connected to the commissioning button
2	-	
3	3.3V	
4	GND	

Grove I2C	Signal	Comments
1	DIO1/I2C_SCL	
2	DIO11/I2C_SDA	
3	3.3V	
4	GND	

Grove PWM0	Signal	Comments
1	RSSI/PWM0	Signal connected to the RSSI LED
2	-	
3	3.3V	
4	GND	

Grove AD2	Signal	Comments
1	AD2	
2	-	
3	3.3V	
4	GND	

SMT board Grove connectors pinout

The following tables show the pinout for the SMT board Grove connectors:

Grove DIO12	Signal	Comments
1	DIO12	
2	-	
3	3.3V	
4	GND	

Grove DIO4	Signal	Comments
1	DIO4	Signal connected to the LED/button
2	-	
3	3.3V	
4	GND	

Grove AD0	Signal	Comments
1	AD0/CB	Signal connected to the commissioning button
2	-	
3	3.3V	
4	GND	

Grove AD3	Signal	Comments
1	AD3	
2	-	
3	3.3V	
4	GND	

Grove I2C	Signal	Comments
1	DIO1/I2C_SCL	
2	DIO11/I2C_SDA	
3	3.3V	
4	GND	

Grove PWM0	Signal	Comments
1	RSSI/PWM0	Signal connected to the RSSI LED
2	-	
3	3.3V	
4	GND	

Grove DIO19	Signal	Comments
1	DIO19	
2	-	
3	3.3V	
4	GND	

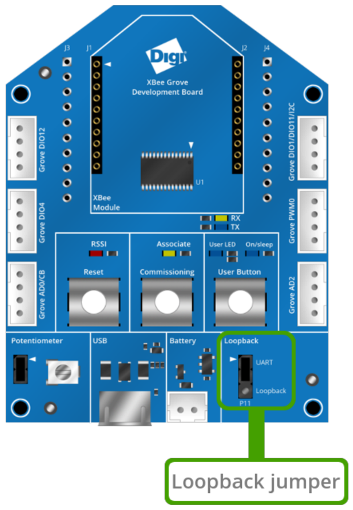
Grove DIO18	Signal	Comments
1	DIO18	
2	-	
3	3.3V	
4	GND	

Loopback jumper

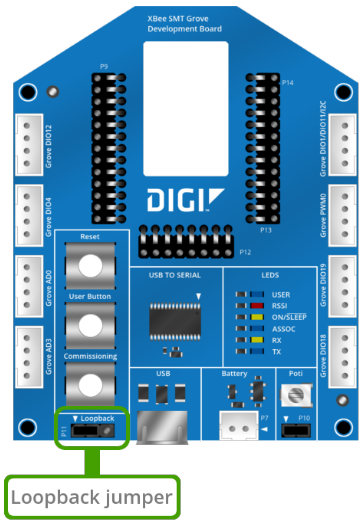
The XBee Grove Development Board provides a three-pin jumper to connect the UART to the USB (normal mode) or to make a loopback connection between the RX and TX signals of the UART.

In loopback mode, connect the RX line to the TX line, which transmits back any data received. You can use loopback in transparent mode to check the signal strength and perform a range test.

XBee THT Grove Development Board Loopback jumper



XBee SMT Grove Development Board Loopback jumper

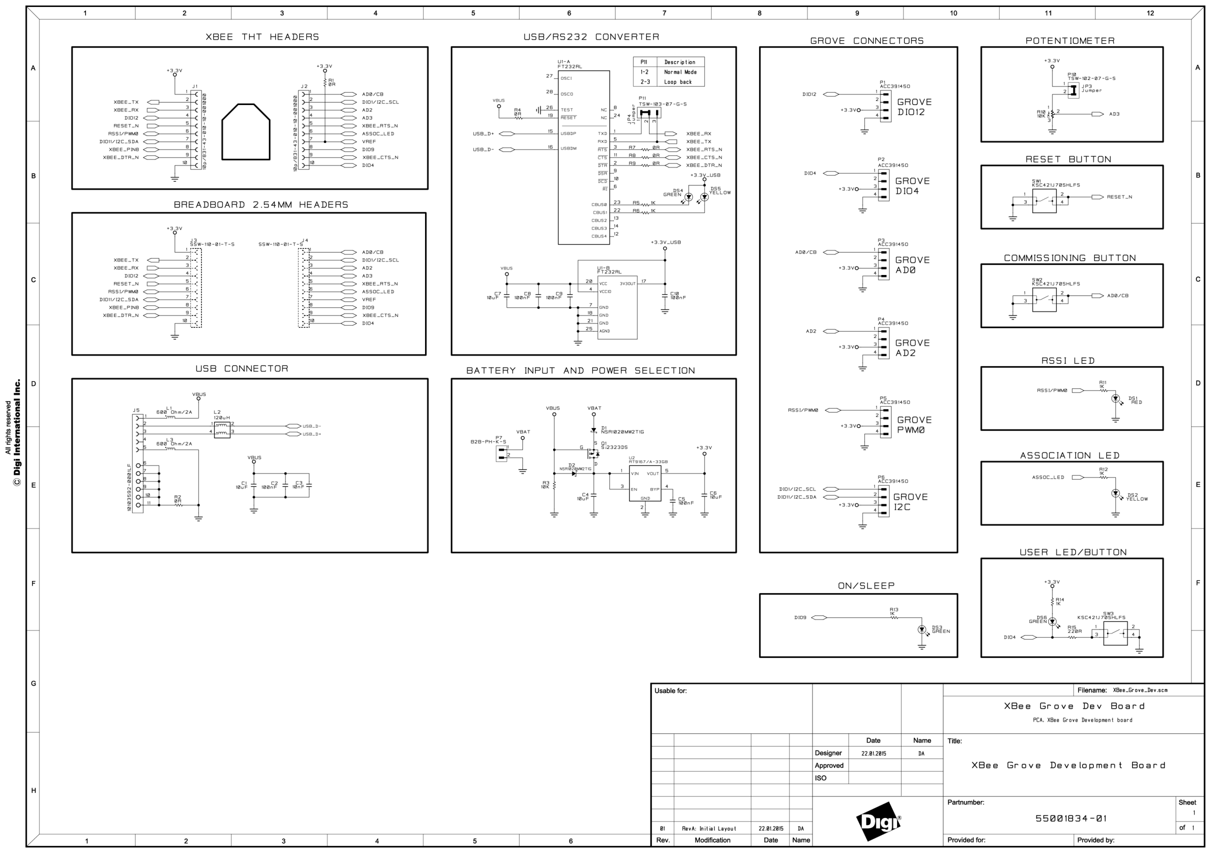


Schematic and Gerber files

This section shows the schematics for the THT Grove Development Board and the SMT Grove Development board and provides links to download the Gerber files.

- XBee THT Grove Development Board
- XBee SMT Grove Development Board

XBee THT Grove Development Board schematic

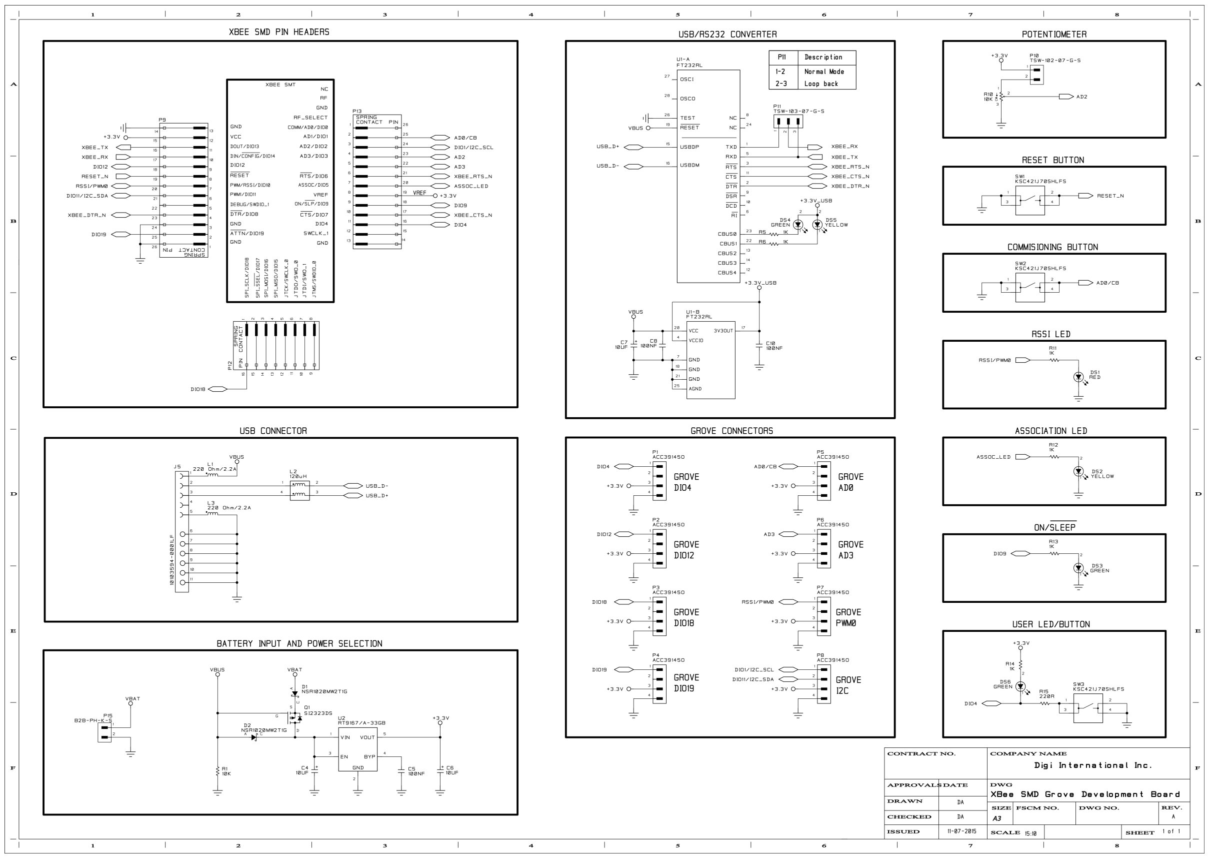


You can dowload a copy of the schematic for the XBee THT Grove Development Board.

Gerber files

You can download the Gerber files for the XBee THT Grove Development Board.

XBee SMT Grove Development Board schematic



You can download a copy of the schematic for the XBee SMT Development Board.

Gerber files

You can download the Gerber files for the XBee SMT Grove Development Board.